EcoStruxure Machine Expert
Generic Libraries

User Guide

07/2020

EIO0000004314.00

www.schneider-electric.com

Table of Contents

1 EcoStruxure Machine Expert - CommonToolbox Library Guide . . . . . Part I

2 EcoStruxure Machine Expert - Data Logging Functions DataLogging - Library Guide. . . . . Part II

3 EcoStruxure Machine Expert - FileFormatUtility Library Guide . . . . . Part III

4 EcoStruxure Machine Expert - PackML Library Guide . . . . . Part IV

5 EcoStruxure Machine Expert - PD_ETest Library Guide . . . . . Part V

6 EcoStruxure Machine Expert - Getting&Setting Real Time Clock SysTimeRtc and SysTimeCore - Library Guide . . . . . Part VI

7 EcoStruxure Machine Expert - Miscellaneous Functions Toolbox Library Guide. . . . . Part VII

8 EcoStruxure Machine Expert - Manage a Cyclic Task Interval Toolbox_Advance - Library Guide. . . . . Part VIII

9 EcoStruxure Machine Expert - Twido Emulation Support Library Library Guide . . . . . Part IX

EcoStruxure Machine Expert
CommonToolbox
Library Guide

06/2020

EIO0000004219.00

www.schneider-electric.com

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

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

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

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

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

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

© 2020 Schneider Electric. All rights reserved.

Table of Contents

Safety Information 5
About the Book 9
Part I General Information 13
Chapter 1 Presentation of the Library 15
General Information 15
Part II Enumerations and Structures 17
Chapter 2 Enumerations 19
ET_Result 19
Part III Global Variables 21
Chapter 3 Global Constants List 23
Global Constants List (GCL) 23
Part IV Program Organization Units (POU) 25
Chapter 4 Function Blocks 27
4.1 FB_RandomGenerator 28
FB_RandomGenerator - General Information 29
FB_RandomGenerator - Init (Method) 31
FB_RandomGenerator - GetDint (Method) 33
FB_RandomGenerator - GetLreal (Method) 35
4.2 FB_RuntimeMeasurement 37
FB_RuntimeMeasurement - General Information 38
FB_RuntimeMeasurement - Reset (Method) 41
FB_RuntimeMeasurement - Start (Method) 42
FB_RuntimeMeasurement - End (Method) 43
Chapter 5 Functions 45
5.1 FC_EtResultToString 46
FC_EtResultToString 46
5.2 FC_LrealToString 47
FC_LrealToString - General Information 47
5.3 FC_MultiConcat 51
FC_MultiConcat - General Information 51

5.4 FC_OpenSlcCommunication 53
FC_OpenSlcCommunication - General Information 53
5.5 FC_CloseSlcCommunication 56
FC_CloseSlcCommunication - General Information 56
Index 59

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge and experience in industrial control programming.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

BEFORE YOU BEGIN

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

⚠ WARNING

UNGUARDED EQUIPMENT

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

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

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

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

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

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

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

START-UP AND TEST

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

⚠ WARNING

EQUIPMENT OPERATION HAZARD

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

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

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

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

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

Before energizing equipment:

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

OPERATION AND ADJUSTMENTS

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

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

About the Book

At a Glance

Document Scope

This document describes the CommonToolbox library.

The library provides functions and function blocks for various tasks, for example, conversion, SLC (Safety Logic Controller) communication, string functions, or application solutions.

Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert V1.2.4.

The technical characteristics of the devices described in this document also appear online. To access this information online, go to the Schneider Electric home page *www.se.com*.

The characteristics that are described in the present document should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the document and online information, use the online information as your reference.

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.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Update your application program as required, paying particular attention to I/O address adjustments, whenever you modify the hardware configuration.

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

Incomplete file transfers, such as data files, application files and/or firmware files, may have serious consequences for your machine or controller. If you remove power, or if there is a power outage or communication interruption during a file transfer, your machine may become inoperative, or your application may attempt to operate on a corrupted data file. If an interruption occurs, reattempt the transfer. Be sure to include in your risk analysis the impact of corrupted data files.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION, DATA LOSS, OR FILE CORRUPTION

- Do not interrupt an ongoing data transfer.
- If the transfer is interrupted for any reason, re-initiate the transfer.
- Do not place your machine into service until the file transfer has completed successfully, unless you have accounted for corrupted files in your risk analysis and have taken appropriate steps to prevent any potentially serious consequences due to unsuccessful file transfers.

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

Related Documents

Document title	Reference
How to Configure the Firewall for PacDrive LMC Controllers, User Guide	EIO0000004198 (ENG) *(see How to Configure the Firewall for PacDrive LMC Controllers, User Guide)* EIO0000004199 (GER) *(see How to Configure the Firewall for PacDrive LMC Controllers, User Guide)*

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

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.

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

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

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

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

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

Part I
General Information

Chapter 1
Presentation of the Library

General Information

Library Overview

The library can, in general, be used with any logic and motion controllers supported by the EcoStruxure Machine Expert software (except for those functions as noted in the present documentation). The library provides functions and function blocks for various tasks, for example, conversion, SLC (Safety Logic Controller) communication, string functions, or application solutions.

Characteristics of the Library

The table indicates the characteristics of the library:

Characteristic	Value
Library title	CommonToolbox
Company	Schneider Electric
Category	• **Application/Util** • **Util**
Component	**Seco - LibrariesEssentials**
Default namespace	**SE_CTBX**
Language model attribute	Qualified-access-only *(see EcoStruxure Machine Expert, Functions and Libraries User Guide)*
Forward compatible library	Yes (FCL)

NOTE: For this library qualified-access-only is set. Therefore, the POUs, data structures, enumerations, and constants have to be accessed using the namespace of the library. The default namespace of the library is **SE_CTBX**.

Part II
Enumerations and Structures

Chapter 2
Enumerations

ET_Result

Overview

Type:	Enumeration
Available as of:	V1.0.0.0

Description

The enumeration ET_Result contains the possible values that indicate the result of operations executed by the POUs of this library.

Enumeration Elements

Name	Data type	Value	Description
Ok	UDINT	0	Operation completed successfully.
StringTooLong	UDINT	1	Concatenated string contains more than 80 characters.
LrealValueTooBig	UDINT	2	LREAL value is too large. Limit = +/- 2000000000
NStartRange	UDINT	20	Value at input i_diNStart is less than or equal to 0.
InitMethodNotCalled	UDINT	21	Initialization method has not yet been executed.
StartMethodNotCalled	UDINT	22	Start method has not yet been executed.
CheckSumOverflow	UDINT	23	Checksum overflow.
FunctionNotSupported	UDINT	100	Function not available for this controller.
InvalidIfSafeLogic	UDINT	200	The interface assigned to i_ifSafeLogic must be provided by an instance of the Safety Logic Controller in the **Devices** tree of the project.

Part III
Global Variables

Chapter 3
Global Constants List

Global Constants List (GCL)

Overview

Type:	Global constants
Available as of:	V1.0.0.0

Description

The Global Constants List (GCL) contains the global constants of the CommonToolbox library.

Global Constants

Variable	Data type	Value	Description
`Gc_sLibraryVersion`	STRING[80]	Vx.x.x.0(1)	Library version
(1) This value varies to indicate the version of the library.			

Part IV
Program Organization Units (POU)

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
4	Function Blocks	27
5	Functions	45

Chapter 4
Function Blocks

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
4.1	FB_RandomGenerator	28
4.2	FB_RuntimeMeasurement	37

Section 4.1
FB_RandomGenerator

What Is in This Section?

This section contains the following topics:

Topic	Page
`FB_RandomGenerator` - General Information	29
`FB_RandomGenerator - Init` (Method)	31
`FB_RandomGenerator - GetDint` (Method)	33
`FB_RandomGenerator - GetLreal` (Method)	35

FB_RandomGenerator - General Information

Overview

Type:	Function block
Available as of:	V1.0.0.0

Task

The function block FB_RandomGenerator generates a pseudo-random number, herein referred to as a random number.

Description

The function block FB_RandomGenerator generates a random number that is within an interval with defined limits upon each call of the GetLreal *(see page 35)* or GetDint *(see page 33)* methods.

The random numbers are generated according to the linear congruence method. The required start value i_diNstart is set using the Init *(see page 31)* method. This can be any integer > 0 which is located within the value range of data type DINT. The Init method has to be called once before the GetLreal or GetDint methods can be called.

The linear congruence method generates a sequence of numbers in accordance with the following regulations:

```
z(i+1) = (a * z(i)) MOD m
```

The variables a and m must be suitable integers. z(0) corresponds to the start value i_diNstart. The random generator uses the values a = 16807 and m = 2^31 - 1 (Mersenne prime number). This generates a sequence of numbers having the period 2^31, that is, the values only repeat after more than 2 billion numbers generated. These may practically be used as random numbers in many computations for specific purposes.

Interface

The function block does not provide any input or output parameters. An instance of the function block must not be called in your application program. To control and monitor, the function block provides methods.

Methods

Name	Description
Init *(see page 31)*	Definition of the default conditions. Has to be called once before calling GetDint or GetLreal.

Name	Description
`GetDint` *(see page 33)*	Generates a random number of the DINT type.
`GetLreal` *(see page 35)*	Generates a random number of the LREAL type.

FB_RandomGenerator - Init (Method)

Overview

Type:	Method
Available as of:	V1.0.0.0
Versions:	Current version



Task

The method `Init` initializes the random generator `FB_RandomGenerator` *(see page 29)*.

Description

The random numbers are generated according to the linear congruence method. The required start value `i_diNstart` is set using the `Init` method. This can be any integer > 0 which is located within the value range of data type DINT.

This method must have been called once before calling the `GetDint` *(see page 33)* or `GetLreal` *(see page 35)* methods.

Interface

Input	Data type	Description
`i_diNStart`	DINT	Start value of the random number row.

Output	Data type	Description
`q_etResult`	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
`q_sResultMsg`	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
BOOL	TRUE= The execution of the method was successful. FALSE = The execution of the method was unsuccessful.

Diagnostic Messages

The following elements of `ET_Result` *(see page 19)* are used for `q_etResult`.

Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.
`NStartRange`	UDINT	20	Value at input `i_diNStart` is less than or equal to 0.

FB_RandomGenerator - GetDint (Method)

Overview

Type:	Method
Available as of:	V1.0.0.0
Versions:	Current version



Task

The method `GetDint` generates a random number.

Description

The method `GetDint` generates a random number of type DINT according to the linear congruence method. Refer to `FB_RandomGenerator` *(see page 29)*.

Interface

Input	Data type	Description
`i_diXmin`	DINT	Lower limit value of the random number.
`i_diXmax`	DINT	Upper limit value of the random number.

Output	Data type	Description
`q_etResult`	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
`q_sResultMsg`	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
DINT	Generated random number of type DINT.

Diagnostic Messages

The following elements of `ET_Result` *(see page 19)* are used for `q_etResult`.

Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.
`InitMethodNotCalled`	UDINT	21	Initialization method has not yet been executed.

FB_RandomGenerator - GetLreal (Method)

Overview

Type:	Method
Available as of:	V1.0.0.0
Versions:	Current version



Task

The method `GetLreal` generates a random number.

Description

The method `GetLreal` generates a random number of type LREAL according to the linear congruence method. Refer to `FB_RandomGenerator` *(see page 29)*.

Interface

Input	Data type	Description
`i_lrXmin`	LREAL	Lower limit value of the random number.
`i_lrXmax`	LREAL	Upper limit value of the random number.

Output	Data type	Description
`q_etResult`	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
`q_sResultMsg`	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
LREAL	Generated random number of type LREAL.

Diagnostic Messages

The following elements of `ET_Result` *(see page 19)* are used for `q_etResult`.

Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.
`InitMethodNotCalled`	UDINT	21	Initialization method has not yet been executed.

Section 4.2
FB_RuntimeMeasurement

What Is in This Section?

This section contains the following topics:

Topic	Page
FB_RuntimeMeasurement - General Information	38
FB_RuntimeMeasurement - Reset (Method)	41
FB_RuntimeMeasurement - Start (Method)	42
FB_RuntimeMeasurement - End (Method)	43

FB_RuntimeMeasurement - General Information

Overview

Type:	Function block
Available as of:	V1.0.0.0

Task

The function block FB_RuntimeMeasurement is used for the runtime measurement of the program code.

Description

In addition to the actual measured time value, it provides the statistical values for minimum, maximum, and mean values.

Interface

The function block does not provide any input or output parameters. An instance of the function block must not be called in your application program. To control and monitor the time measurement, the function block provides methods and properties.

Methods

Name	Description
Reset *(see page 41)*	Resets the function block to its initial state.
Start *(see page 42)*	Starts a new measurement.
End *(see page 43)*	Completes a presently running time measurement.

Properties

Name	Data type	Accessing	Description
udiActTime	UDINT	Read	Duration of the last measurement. Unit: ns
udiAverageTime	UDINT	Read	Mean duration of the measurements performed since the last reset. Unit: ns
udiCycleCount	UDINT	Read	Number of measurements performed since the last reset.
udiMaxTime	UDINT	Read	Longest measured period since the last reset. Unit: ns

Name	Data type	Accessing	Description
udiMinTime	UDINT	Read	Shortest measured period since the last reset. Unit: ns

Code Example

Declaration

```
PROGRAM RunTimeMeasurement
VAR
     // Instance of the function block
     fbRunTimeMeasurement    : SE_CTBX.FB_RunTimeMeasurement;

     // Measured variables
     udiActTime              : UDINT;
     udiAverageTime          : UDINT;
     udiCycleCount           : UDINT;
     udiMaxTime              : UDINT;
     udiMinTime              : UDINT;

     // Auxiliary variables
     uiLoopCtr               : UINT;
     uiCycleCtr              : UINT;
     xInit                   : BOOL;
END_VAR
```

Program code

```
IF NOT xInit THEN
     // Reset the function block to its initial state
     fbRunTimeMeasurement.Reset();

     xInit                   := TRUE;
     uiCycleCtr              := 0;
END_IF

// Start a new measurement
fbRunTimeMeasurement.Start();

// Code execution
WHILE uiLoopCtr < 100 DO
      uiLoopCtr := uiLoopCtr + 1;
END_WHILE

// Complete a currently running time measurement
fbRunTimeMeasurement.End();
```

```
// Get measured values after 10 program cycles
uiCycleCtr := uiCycleCtr + 1;

IF uiCycleCtr = 10 THEN
    // Read out the measured values
    udiActTime              := fbRunTimeMeasurement.udiActTime;
    udiAverageTime          := fbRunTimeMeasurement.udiAverageTime;
    udiCycleCount           := fbRunTimeMeasurement.udiCycleCount;
    udiMaxTime              := fbRunTimeMeasurement.udiMaxTime;
    udiMinTime              := fbRunTimeMeasurement.udiMinTime;

    // Restart measurement
    xInit := FALSE;
END_IF
```

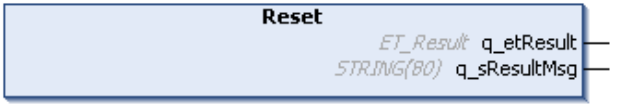
Measured values with an M251 controller (MAST cycle time of 10 ms)

```
// Read out the measured values
udiActTime      7000          := fbRunTimeMeasurement.udiActTime;
udiAverageTime  7600          := fbRunTimeMeasurement.udiAverageTime;
udiCycleCount   10            := fbRunTimeMeasurement.udiCycleCount;
udiMaxTime      9000          := fbRunTimeMeasurement.udiMaxTime;
udiMinTime      7000          := fbRunTimeMeasurement.udiMinTime;
```

FB_RuntimeMeasurement - Reset (Method)

Overview

Type:	Method
Available as of:	V1.0.0.0
Versions:	Current version



Task

The method `Reset` resets the function block to its initial state.

Description

The measured values and statistical data are reset to 0.

Interface

Output	Data type	Description
`q_etResult`	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
`q_sResultMsg`	STRING [80]	Provides additional diagnostic and status information as a text message.

Diagnostic Messages

The following element of `ET_Result` *(see page 19)* is used for `q_etResult`.

Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.

FB_RuntimeMeasurement - Start (Method)

Overview

Type:	Method
Available as of:	V1.0.0.0
Versions:	Current version



Task

The method `Start` starts a new measurement.

Description

The starting time of the next measurement is recorded. By calling `FB_RunTimeMeasurement.End` *(see page 43)* after the code block to be examined, the measurement is evaluated.

Interface

Output	Data type	Description
`q_etResult`	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
`q_sResultMsg`	STRING [80]	Provides additional diagnostic and status information as a text message.

Diagnostic Messages

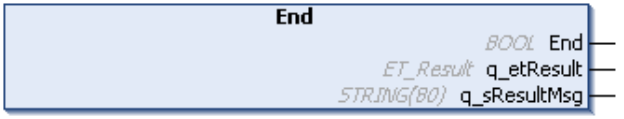
The following element of `ET_Result` *(see page 19)* is used for `q_etResult`.

Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.

FB_RuntimeMeasurement - End (Method)

Overview

Type:	Method
Available as of:	V1.0.0.0
Versions:	Current version



Task

The method End completes a presently running time measurement.

Description

The presently running time measurement is being completed and evaluated. The statistical data of the function block are being updated.

Interface

Output	Data type	Description
q_etResult	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
q_sResultMsg	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
BOOL	Indicates whether or not the execution of the method was successful.

Diagnostic Messages

The following elements of ET_Result *(see page 19)* are used for q_etResult.

Name	Data type	Value	Description
Ok	UDINT	0	Operation completed successfully.

Name	Data type	Value	Description
StartMethodNotCalled	UDINT	22	Start method has not yet been executed.
CheckSumOverflow	UDINT	23	Checksum overflow.

Chapter 5
Functions

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
5.1	FC_EtResultToString	46
5.2	FC_LrealToString	47
5.3	FC_MultiConcat	51
5.4	FC_OpenSlcCommunication	53
5.5	FC_CloseSlcCommunication	56

Section 5.1
FC_EtResultToString

FC_EtResultToString

Overview

Type:	Function
Available as of:	V1.0.0.0
Inherits from:	-
Implements:	-



Task

Convert an enumeration element of type ET_Result to a variable of type STRING.

Functional Description

Using the function `FC_EtResultToString`, you can convert an enumeration element of type ET_Result to a variable of type STRING.

Interface

Input	Data type	Description
`i_etResult`	ET_Result	Enumeration with the result.

Return Value

Data type	Description
STRING(80)	The ET_Result converted to text. If `i_etResult` is indeterminable the return value is: `Unknown Result: <Value of the input i_etResult>`

Section 5.2
FC_LrealToString

FC_LrealToString - General Information

Overview

Type:	Function
Available as of:	V1.0.0.0
Versions:	Current version



Task

The function FC_LrealToString converts numerical values into a formatted string.

Description

The function converts any numerical value to a STRING with freely defined format.

```
i_sFormat = '[B][S][PPPP][.][FFFF][A][aa]'
```

Numerical value (i_lrInput)	Defined format (i_sFormat)	Result STRING[80]
12.3456	‘10#00.000000L15’	‘12.345600 ‘ (6 spaces at the right-hand side)

The format is specified in the input variable `i_sFormat` as follows:

Placeholder	Description
`[B]`	Sets the base of the number representation: ● `'2'` = binary For example, when `'2#'` is entered, a decimal value of 30 is converted to `'11110'`. ● `'8'` = octal For example, when `'8#'` is entered, a decimal value of 30 is converted to `'36'`. ● `'10'` = decimal ● `'16'` = hexadecimal For example, when `'16#'` is entered, a decimal value of 30 is converted to `'1E'`. If no value is given for `'B'`, `'10'` is used as default value.
`[S]`	Represent the mathematical sign. If a base is entered, then the mathematical sign representation must also be specified. Otherwise, this entry is dispensed with and `'#'` is used as default. As a basic principle, a minus sign precedes negative values, independent of the format specification. In other respects, the following options exist for formatting: ● `'#'`: Nothing precedes non-negative numbers. For example, when `'10#'` is entered, a decimal value of 123 is converted to `'123'`. A negative decimal value -123 stays as is `'-123'`. ● `'-'`: An empty space precedes non-negative numbers. For example, when `'10-'` is entered, a decimal value of 123 is converted to `'[space]123'`. A negative decimal value -123 stays as is `'-123'`. ● `'+'`: A '+' (plus sign) precedes positive numbers, an empty space precedes 0. For example, when `'10+'` is entered, a decimal value of 123 is converted to `'+123'`. A negative decimal value -123 stays as is `'-123'`.
`[PPPP]`	Each `'P'` character before the decimal point represents a place preceding the decimal point in the output. This is used to set the minimum number of characters that are reserved for the whole number component. Additionally this defines the content for the output when the numerical representation is shorter than the format string expects. If `'P'` is a space, then a space is inserted at that position. If `'P'` is zero, then a zero is inserted. Using this portion of the format input you can implement leading zeros or, in other words, you can set the decimal point to a fixed position from the left edge of the output string. If only a zero is specified, all places preceding the decimal point of the transferred LREAL are converted to a string. For example: ● When `'10#000000'` is entered, a decimal value of 123 is converted to `'000123'`. ● When `'10#[6 spaces'` is entered, a decimal value of 123 is converted to `'[3 spaces]123'`. ● When `'10+0000'` is entered, a decimal value of 123 is converted to `'+0123'`.
`[.]`	The decimal point is only required when fractional characters are desired.

Placeholder	Description
[FFFF]	The 'F' signs following the decimal point determine the number of spaces following the decimal point in the output. Each 'F' represents one position following the decimal point. If the input value has more positions following the decimal point than the format provides for, the remainder is deleted. If it has fewer, then it is filled up according to 'F'. If 'F' is an empty space, then an empty space is put into that position. If 'F' is zero, then a zero is put there. If 'F' is an underline, nothing is entered. This input is used to either fix the number of positions after the decimal point or to determine the distance of the decimal point from the right edge. For example: ● When '10#.0' is entered, a decimal value of 123.456 is converted to '123.4'. ● When '10+.000' is entered, a decimal value of 123.456 is converted to '+123.456'. ● When '10-.0000' is entered, a decimal value of 123.456 is converted to '[space]123.4560'. ● When '10-.00' is entered, a decimal value of -123.456 is converted to '-123.45'. ● When '10-0000.000' is entered, a decimal value of -123.456 is converted to '-0123.456'.
[A] [aa]	'A' an 'aa' determine if the output has a fixed total length and if it is to be left or right justified or centered. Possible values are 'L' for left justified, 'R' for right justified; or 'C' for centered. The desired length of the output is determined by 'aa'. If the output has more characters than specified, the value to the right is filled out with empty spaces so that the numbers are displayed according to 'A'. If no adjustment is desired, simply do not specify any 'A' or 'aa' in the format string. But if one of the two values is specified, then the other must be specified as well. For example: ● When '10#.000L20' is entered, a decimal value of 123.456 is converted to '123.456[13 spaces]'. ● When '10+.000R20' is entered, a decimal value of 123.456 is converted to '[12 spaces]+123.456'. ● When '10#.00C20' is entered, a decimal value of 123.456 is converted to '[7 spaces]123.46[7 spaces]'.

Interface

Input	Data type	Description
i_sFormat	STRING[80]	Formatting the output.
i_lrInput	LREAL	The value to be formatted.

Output	Data type	Description
q_etResult	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
q_sResultMsg	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
STRING[80]	The value transferred at `i_lrInput` is converted to a STRING in the specified format and returned.

Diagnostic Messages

The following elements of `ET_Result` *(see page 19)* are used for `q_etResult`.

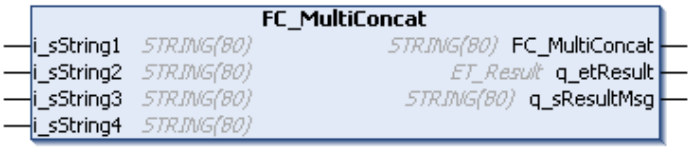
Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.
`LrealValueTooBig`	UDINT	2	LREAL value is too large. Limit = +/- 2000000000

Section 5.3
FC_MultiConcat

FC_MultiConcat - General Information

Overview

Type:	Function
Available as of:	V1.0.0.0
Versions:	Current version



Task

The function FC_MultiConcat concatenates four given strings to a single, composite string.

Description

The function concatenates the four input strings according to their sequence to a composite STRING with a maximum length of 80 character.

Interface

Input	Data type	Description
i_sString1	STRING[80]	String 1
i_sString2	STRING[80]	String 2
i_sString3	STRING[80]	String 3
i_sString4	STRING[80]	String 4

Output	Data type	Description
q_etResult	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
q_sResultMsg	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
STRING[80]	The composite string.

Diagnostic Messages

The following elements of ET_Result *(see page 19)* are used for q_etResult.

Name	Data type	Value	Description
Ok	UDINT	0	Operation completed successfully.
StringTooLong	UDINT	1	Concatenated string contains more than 80 characters.

Section 5.4
FC_OpenSlcCommunication

FC_OpenSlcCommunication - General Information

Overview

Type:	Function
Available as of:	V1.0.0.0
Versions:	Current version



Description

NOTE: This function is only supported by PacDrive LMC controllers.

The function `FC_OpenSlcCommunication` activates the required port rules for the controller firewall. This allows communication of the programming software with the SLC (Safety Logic Controller) via the controller.

- After a restart, the firewall is configured according to the default configuration file. Refer to *How to Configure the Firewall for PacDrive LMC Controllers User Guide (see How to Configure the Firewall for PacDrive LMC Controllers, User Guide)*.
- It is a good practice to keep the port rules activated only as long as needed. Use the `FC_CloseSlcCommunication` *(see page 56)* to deactivate the port rules after your communication to the SLC (Safety Logic Controller) is no longer needed.

⚠ WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

- Evaluate whether your environment or your machines are connected to your critical infrastructure and, if so, take appropriate steps in terms of prevention, based on Defense-in-Depth, before connecting the automation system to any network.
- Limit the number of devices connected to a network to the minimum necessary.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- Prepare a recovery plan including backup of your system and process information.

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

Interface

Input	Data type	Description
`i_ifSafeLogic`	SystemConfigurationItf.IF_SafeLogic	Specifies the SLC (Safety Logic Controller) for which the communication should be enabled.

Output	Data type	Description
`q_etResult`	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
`q_sResultMsg`	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
BOOL	TRUE = SLC connection settings were adapted successfully. FALSE = SLC connecting settings cannot be set.

Diagnostic Messages

The following elements of `ET_Result` *(see page 19)* are used for `q_etResult`.

Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.
`FunctionNotSupported`	UDINT	100	Function not available for this controller.
`InvalidIfSafeLogic`	UDINT	200	The interface assigned to `i_ifSafeLogic` must be provided by an instance of the Safety Logic Controller in the **Devices** tree of the project.

Section 5.5
FC_CloseSlcCommunication

FC_CloseSlcCommunication - General Information

Overview

Type:	Function
Available as of:	V1.0.0.0
Versions:	Current version



Description

NOTE: This function is only supported by PacDrive LMC controllers.

The function `FC_CloseSlcCommunication` deactivates the port rules for the controller firewall. (Also refer to `FC_OpenSlcCommunication` *(see page 53)*).

Interface

Input	Data type	Description
`i_ifSafeLogic`	SystemConfigurationItf.IF_SafeLogic	Specifies the SLC (Safety Logic Controller) for which the communication should be disabled.

Output	Data type	Description
`q_etResult`	ET_Result *(see page 19)*	Provides diagnostic and status information as an enumeration value.
`q_sResultMsg`	STRING [80]	Provides additional diagnostic and status information as a text message.

Return Value

Data type	Description
BOOL	Indicates whether or not the execution of the method was successful.

Diagnostic Messages

The following elements of `ET_Result` *(see page 19)* are used for `q_etResult`.

Name	Data type	Value	Description
`Ok`	UDINT	0	Operation completed successfully.
`FunctionNotSupported`	UDINT	100	Function not available for this controller.
`InvalidIfSafeLogic`	UDINT	200	The interface assigned to `i_ifSafeLogic` must be provided by an instance of the Safety Logic Controller in the **Devices** tree of the project.

Index

C

CommonToolbox, *15*
- FB_RandomGenerator, *29*
- FB_RuntimeMeasurement, *38*
- FC_CloseSlcCommunication , *56*
- FC_LrealToString, *47*
- FC_MultiConcat, *51*
- FC_OpenSlcCommunication , *53*
- GCL (Global Constants List), *23*

E

End, *43*
ET_Result, *19*
- CheckSumOverflow, *19*, *44*
- FunctionNotSupported, *19*, *55*, *57*
- InitMethodNotCalled, *19*, *34*, *36*
- InvalidIfSafeLogic, *19*, *55*, *57*
- LrealValueTooBig, *19*, *50*
- NStartRange, *19*, *32*
- Ok, *19*, *32*, *34*, *36*, *41*, *42*, *43*, *50*, *52*, *55*, *57*
- StartMethodNotCalled, *19*, *44*
- StringTooLong, *19*, *52*

F

FB_RandomGenerator, *29*
- GetDint, *33*
- GetLreal, *35*
- Init, *31*

FB_RuntimeMeasurement, *38*
- End, *43*
- Reset, *41*
- Start, *42*

FC_CloseSlcCommunication , *56*
FC_EtResultToString, *46*
FC_LrealToString, *47*
FC_MultiConcat, *51*
FC_OpenSlcCommunication , *53*

G

GCL (Global Constants List)
- CommonToolbox, *23*

GetDint, *33*
GetLreal, *35*

I

Init, *31*

R

Reset, *41*

S

Start, *42*

EcoStruxure Machine Expert

Data Logging Functions DataLogging Library Guide

06/2019

EIO0000002938.00

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

Safety Information 5
About the Book 7
Chapter 1 Data Logging 11
Introduction to Data Logging 12
Configuring the Data Log 14
Managing Data Log Files 16
Additional Data Log File Information 20
Adding LogRecord and Dump Function Blocks 21
Building a Wide WSTRING 22
Appendices 23
Appendix A Function and Function Block Representation 25
Differences Between a Function and a Function Block 26
How to Use a Function or a Function Block in IL Language 27
How to Use a Function or a Function Block in ST Language 30
Glossary 33
Index 35

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

About the Book

At a Glance

Document Scope

This guide explains the data logging functionality for controllers that support file management operations.

Validity Note

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

Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2015	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems

Standard	Description
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2016	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

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

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

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

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

Chapter 1
Data Logging

Introduction

This document explains the data logging functionality for controllers that support file management operations.

A data log is a text file of user-defined strings that contain application data information for a process or machine. The data log file is stored on the controller. You can upload the file and open it with a standard text editor. The information includes embedded variable values and their associated text. Time and date stamping are additional options.

What Is in This Chapter?

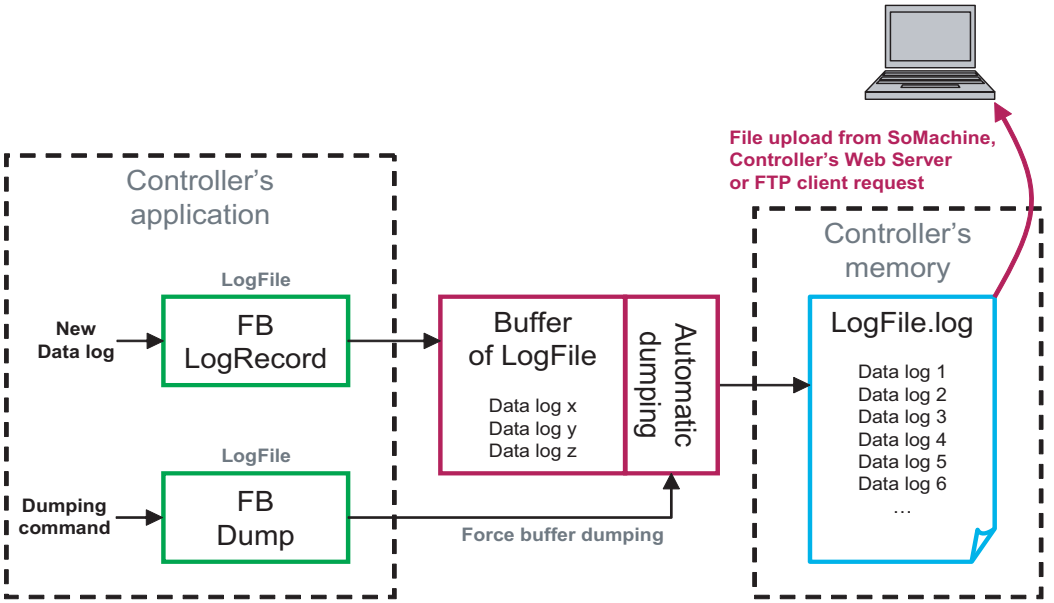
This chapter contains the following topics:

Topic	Page
Introduction to Data Logging	12
Configuring the Data Log	14
Managing Data Log Files	16
Additional Data Log File Information	20
Adding LogRecord and Dump Function Blocks	21
Building a Wide WSTRING	22

Introduction to Data Logging

Overview

You can monitor and analyze application data by examining the data log file (.log).



The figure shows an application that includes the 2 function blocks, `LogRecord` and `Dump`. The `LogRecord` function block writes data to the buffer, which empties into the data log file (.log) located into the controller memory. The buffer dumping is automatic when 80% full or it can be forced by the `Dump` function. As a standard FTP client, a PC can access this data log file when the controller acts as an FTP server. It is also possible to upload the file with EcoStruxure Machine Expert or by the controller web server.

NOTE: Only controllers with file management functionality can support data logging. Refer to your controller programming manual to see if it supports file management. The software itself does not evaluate your controller for compatibility with data logging activities.

Sample Data Log File (.log)

```
Entries in File: 8; Last Entry: 8;
18/06/2009;14:12:33;cycle: 1182;
18/06/2009;14:12:35;cycle: 1292;
18/06/2009;14:12:38;cycle: 1450;
18/06/2009;14:12:40;cycle: 1514;
18/06/2009;14:12:41;cycle: 1585;
18/06/2009;14:12:43;cycle: 1656;
18/06/2009;14:14:20;cycle: 6346;
18/06/2009;14:14:26;cycle: 6636;
```

Implementation Procedure

First declare and configure the data log files in your application before starting to write your program.

Configuring the Data Log

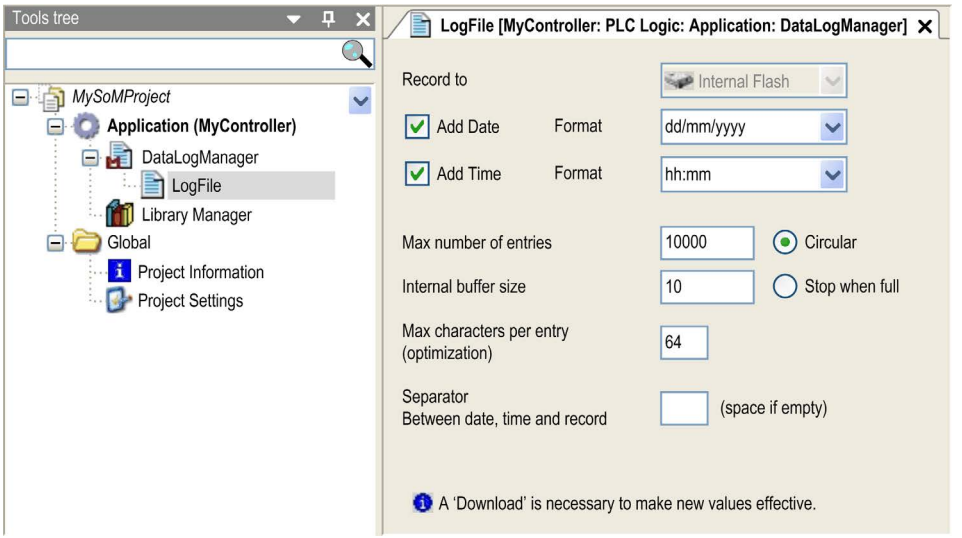
Add a Data Log Manager

Add a data log manager to your configuration before configuring data logging:

Step	Action
1	In the **Tools tree**, select the **Application** node, click the green plus sign and select **Add other objects → DataLogManager...**. **Result:** The **Add Data Log Manager** dialog box appears.
2	In the **Add Data Log Manager** dialog box, click **Add**. **Result:** A **DataLogManager** node appears below the **Application** node.
3	Select the **DataLogManager** node, click the green plus sign, and select **DataLog...**. **Result:** The **Add DataLog** dialog box appears.
4	In the **Data Logging File Name** text box, enter a name for your data log file, and click **Add**. **Result:** The data log file with the given name appears below the **DataLogManager** node, and the configuration screen opens in the editor view in the middle of the EcoStruxure Machine Expert Logic Builder screen. **Note:** The data log file name cannot be changed later.
5	Set the data log file parameters *(see page 14)*.
6	Repeat steps 3 to 5 to create additional data log files.

Configuration Screen

This configuration screen appears after you add a data log file to your configuration:



Configuration Parameters

Parameter	Description	
Add Date **Add Time**	These options print the respective date or time for each record. For example, an instance associated with 10 June 2009 at 2:30 p.m. could be represented as `10/06/2009` or `06/10/2009` or `20090610` ... at `14:30` or `02:30:00 pm`, etc.	
Max number of entries	This option sets the maximum number of records contained in the data log file. Valid values are from 10 to 65536. (The default is 10000.)	
Mode	**Circular** (default)	When the `Max number of entries` is reached, new records overwrite old records. The first line of the data log file can be used to get the actual position of the last record as well as the rank of other records. The following cases are based on a file with a 10-record maximum: • **Case 1:** `Entries in File: 8; Last Entry: 8` **Meaning:** The number of entries in the file equals the value of `Last Entry`. The records are ranked from N°1 (the oldest) to N°8 (the most recent). • **Case 2:** `Entries in File: 10; Last Entry: 5` **Meaning:** The number of entries in the file is greater than the value of `Last Entry`. The file is full and a new record replaces the oldest one in the file. The 10 records are ranked in this order (oldest to newest): 6, 7, 8, 9, 10, 1, 2, 3, 4, 5
	Stop when full	When the `Max number of entries` is reached, a new record attempt returns a detected error message.
Internal Buffer Size	Configure the size of the RAM buffer that stores the added records. Valid values are from 1 to the value configured in **Max number of entries**. (The default is 10.)	
Max characters per entry (optimization)	Set the maximum length of each entry. Valid values are from 10 to 255. (The default is 64.) The maximum length includes separators and optional date and time. Additional spaces are added at the end of the line to accommodate the number of defined characters.	
Separator between date, time and record	Define the character to be inserted between different fields in the data log. An additional separator is added after the record.	
NOTE: For each configured data log file, there is a `LogRecord` function block instance with the same name as this data log file that handles all internal data and allows data logging management.		

Managing Data Log Files

Introduction

One function block, `LogRecord`, is provided for writing text string entries to the data log file. This function block stores the input string in an internal buffer. When this buffer reaches 80 % of capacity, it is moved to the real file on the controller. You can force this save mechanism by using the `Dump` function block.

When power is removed, you may lose the data in the internal buffer or increase the cycle time until the buffer is emptied.

NOTICE
LOSS OF DATA • Do not remove power to the controller until all internal buffer information has been moved to the actual file system. • If the data being recorded is important to your application, configure your internal buffer size to 1. **Failure to follow these instructions can result in equipment damage.**

Adding a Record with the LogRecord Function Block

This function block is available for logging a UNICODE string in a specific log:



Input Parameters

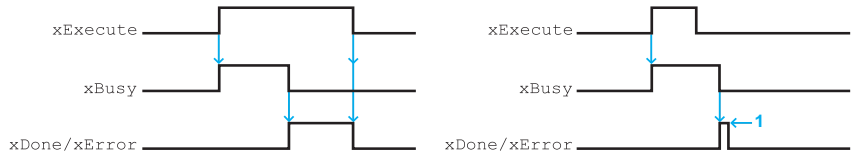
Parameter	Type	Comment
`xExecute`	`BOOL`	The function is executed on the rising edge of this input. **NOTE:** When `xExecute` is set to TRUE at the first task cycle in RUN after a cold or warm reset, the rising edge is not detected.
`wsRecord`	`WSTRING`	This user-specified UNICODE text string is written to the data log file. **NOTE:** The WSTRING *(see page 22)* type is available in the `Standard64.lib` library that is automatically inserted when the data log manager is added to the application.

Output Parameters

Parameter	Type	Comment
xDone	BOOL	This output is set to TRUE when the record is successfully saved to the internal buffer without error messages.
xBusy	BOOL	This output remains TRUE while LogRecord is busy (until the transfer to the buffer is complete).
xError	BOOL	This output is set to TRUE when an error occurs (for example, when the internal buffer is full).
eError	ERROR	This output contains the error code when xError is TRUE: • NO_ERROR • INIT_ERROR • DUMP_ERROR • BUFFER_FULL_ERROR • FILE_FULL_ERROR • DUMP_INCOMPLETED • INPUT_ERROR • FILE_OPEN_ERROR • FILE_SETPOINTER_ERROR • FILE_WRITE_ERROR • FILE_CLOSE_ERROR

NOTE: If the record exceeds the configured length, it is truncated.

The xDone and xError outputs remain TRUE as long as xExecute is TRUE. When xExecute is set to FALSE before xDone or xError is set to TRUE (xBusy is still TRUE), one of each is set to TRUE when the function block is completed during one controller cycle so that the application detects this end:



1 One cycle as Ex is FALSE.

The LogRecord function block includes an instance that corresponds to each configured data log file.

NOTE: Do not explicitly declare an instance of the function block because the instance is declared automatically. If you explicitly declare an instance of the function block, a message is displayed (**2 time variable declaration**) and the function block is rendered inoperable.

Add the function block to your POU and specify the appropriate data log file instance with input help (see Adding LogRecord *(see page 21)*).

Forcing a Save with the Dump Method

You can use the `Dump` method (a method can be regarded as sub-function of a function block, in this case `LogRecord`) to force the application data to move stored records in the internal buffer to the real file in the controller's file system:



Parameters

Input	Type	Comment
xExecute	BOOL	The save function is executed on the rising edge of this input. **NOTE:** When `xExecute` is set to 1 at first controller task cycle, the rising edge is not detected.

Output	Type	Comment
xDone	BOOL	This output is set to TRUE when the records are successfully saved without error messages.
xBusy	BOOL	This output remains TRUE while Dump is busy (until the file is completely written).
xError	BOOL	This output is set to TRUE when an error is detected (for example, when the data log file is full).
eError	ERROR	This output contains the error code when `xError` is TRUE: • NO_ERROR • INIT_ERROR • DUMP_ERROR • BUFFER_FULL_ERROR • FILE_FULL_ERROR • DUMP_INCOMPLETED • INPUT_ERROR • FILE_OPEN_ERROR • FILE_SETPOINTER_ERROR • FILE_WRITE_ERROR • FILE_CLOSE_ERROR

Refer to Adding a Dump Method *(see page 21)* for the `Dump` method implementation.

Automatic Save to Data Log File

When the `LogRecord` function block is used, the system automatically moves the data from the internal buffer to the data log file when this buffer is 80 % full. The `Dump` function block allows you to force this move before reaching the 80 %. This 80 % limit allows the write process to begin before the buffer is full and while a new record is added.

This table shows the number of records to which the file is stored for a given configured buffer size (80% of buffer size rounded down to the nearest integer):

Buffer Size	80 % Limit	Comment
1	1	The save to the data log file is activated as soon as a record is added with an additional slot available for a new record that arrives during this save.
2	1	The save to the data log file is activated when the buffer reaches 80% with an additional slot available for a new record that arrives during this save.
3	2	
4	3	
5	4	The save to the data log file is activated when the buffer reaches 80% with additional slots available for new records that arrive during this save.
6	4	
7	5	
8	6	
...		

In the case of either an explicit (`Dump`) or automatic (80 %) save, the data log file closes after each record (or record group) in case there is a subsequent external power failure.

Additional Data Log File Information

Data Log Properties

The data log properties can be accessed after the function block is configured.

The **LogRecord** properties are additional variables (read only) automatically attached to the **LogRecord** instance that provide information about the data log file status:

Variables	Type	Description
<Data Log File name>.NumberOfRecords	`UDINT`	the number of records in the data log file
<Data Log File name>.NumberOfBufferedRecords	`UINT`	the number of records in the buffer
<Data Log File name>.FileStatus	`FILE_STATUS`	status information about the data log file (`FILE_STATUS` type): • 0: OK • 1: FILE_FULL • 2: NO_WRITE_ACCESS • 3: FILE_NOT_EXISTS
<Data Log File name>.DumpInProgress	`BOOL`	TRUE when the buffered records are being saved in the data log file

Recommendation

The **LogRecord** function block needs much more than 15 interval cycles after its activation (with `xExecute`) to save a record in the logfile. Therefore, it is fort recommended to use this function block within a FAST task:

Task Type	Interval (ms)	Minimum Time Needed to Save Record
cyclic	20	300 ms
cyclic	1	15 ms: Fort Recommended
event		needs 15 events: Not Recommended

Adding LogRecord and Dump Function Blocks

Overview

Follow these steps to add the `LogRecord` function block and `Dump` method to your project.

Adding LogRecord

Add the `LogRecord` function block to your project:

Step	Action	Comment
1	Insert a `LogRecord` function block in your POU using the **Input Assistant** or by directly typing **LogRecord**.	From the **Input Assistant** dialog box, make these selections: ● Categories: **Function Blocks (Libraries)** ● Items: **{} SEDL → LogRecord** (Items in structured view)
2	Click **OK** or press **ENTER**.	The `LogRecord` function block is now part of your project.
3	Select the appropriate data log file as **LogRecord** instance name	From the input assistant, select the appropriate data log file or directly enter the data log file name.
4	Configure the inputs and outputs as you would for other function blocks.	Refer to the description of this function block's parameters *(see page 16)*.

Adding the Dump Method

Add the `Dump` method to your project:

Step	Action	Comment
1	Insert a `Dump` method to your POU using the **Input Assistant** or by directly typing **LogRecord.Dump**.	From the **Input Assistant** dialog box, make these selections: ● Categories: **Function Blocks (Libraries)** ● Items: **SEDL → LogRecord → Dump** (Items in structured view)
2	Click **OK** or press **ENTER**.	The **LogRecord.Dump** method is now part of your project.
3	Select the appropriate data log file as **LogRecord** instance name	From the **Input Assistant**, select the appropriate data log file or directly enter the data log file name.
4	Configure the inputs and outputs as you would for other functions.	Refer to the description of this function block's parameters *(see page 18)*.

Building a Wide WSTRING

Overview

The `wsRecord` input of the `LogRecord` function block is of the type `WSTRING` (wide string). To build the log string, you must first add the `Standard64` library to your application and use wide string functions.

Example

This figure shows the creation of a sample `WSTRING` that includes a variable value:

```
PROGRAM POU_1
VAR
    cycle: INT;
    str: WSTRING;
END_VAR

cycle := cycle + 1;
str := wconcat("cycle: ", INT_TO_WSTRING(cycle));
```

Appendices



Appendix A
Function and Function Block Representation

Overview

Each function can be represented in the following languages:

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

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

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Differences Between a Function and a Function Block	26
How to Use a Function or a Function Block in IL Language	27
How to Use a Function or a Function Block in ST Language	30

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.

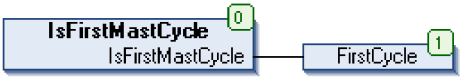
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

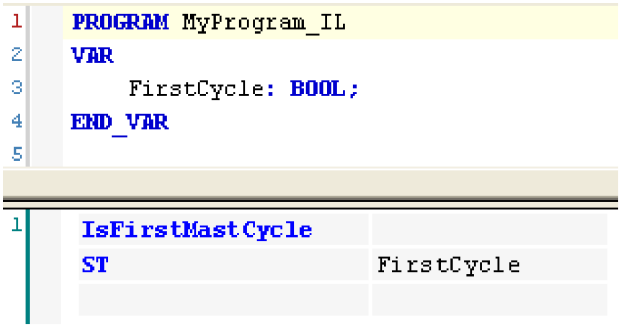
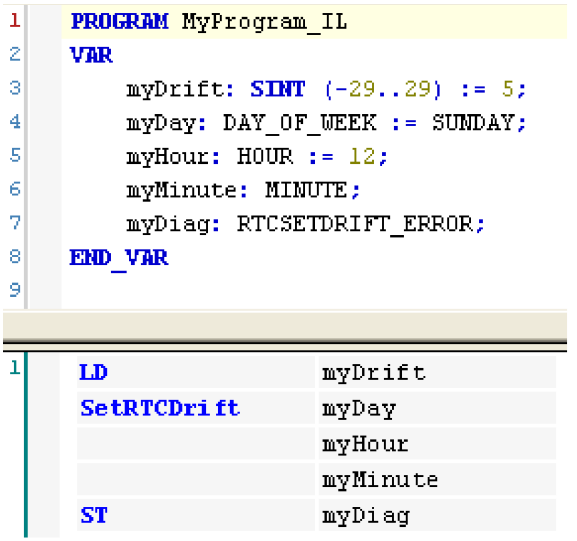
This procedure describes how to insert a function in IL language:

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 EcoStruxure Machine Expert, 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`	
IL example of a function with input parameters: `SetRTCDrift`	

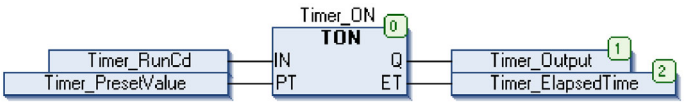
Using a Function Block in IL Language

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

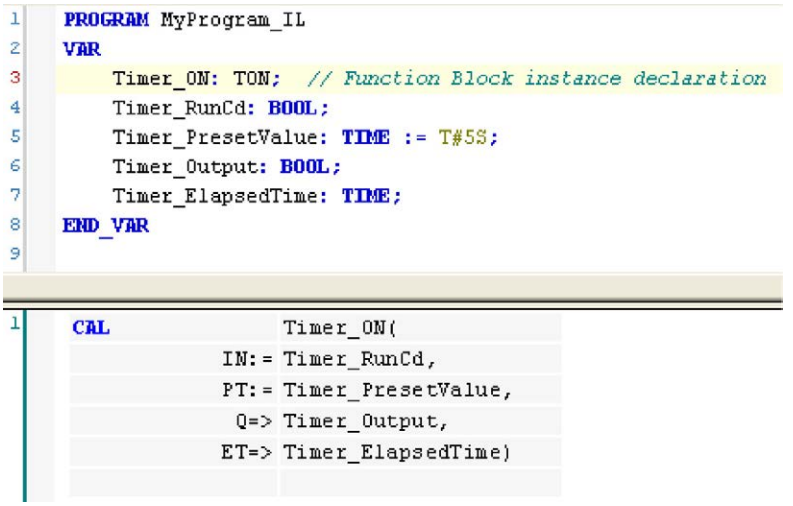
Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see EcoStruxure Machine Expert, Programming Guide)*.

Step	Action
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	

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 EcoStruxure Machine Expert, 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:

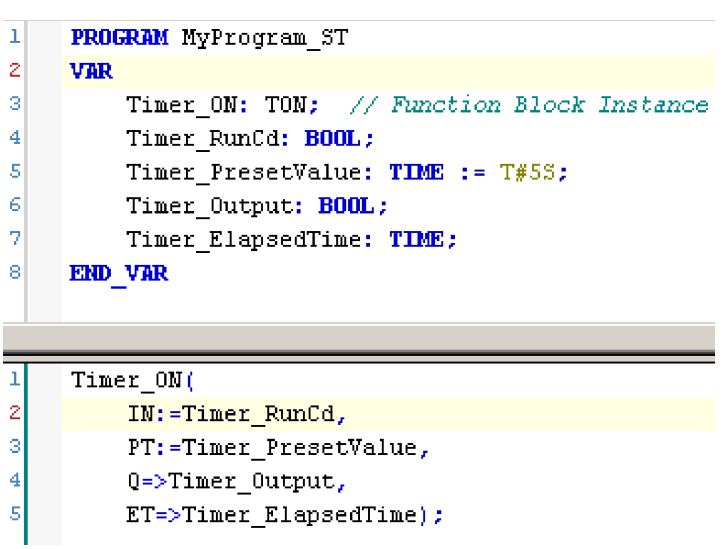
Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation *(see EcoStruxure Machine Expert, Programming Guide)*.
2	Create the input and 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);
```

Glossary

B

byte

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

C

CFC

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

F

FB

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

function block diagram

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

I

IL

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

INT

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

L

LD

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

P

POU

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

S

ST

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

V

variable

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

Index

D

data logging, *11*
Dump
    Function Block, *18*
DumpInProgress
    Variables, *20*

E

ERROR
    Output Parameter Type, *17*, *18*

F

FILE_STATUS
    Type, *20*
FileStatus
    Variables, *20*
Function Block
    Dump, *18*
    LogRecord, *16*
functions
    differences between a function and a function block, *26*
    how to use a function or a function block in IL language, *27*
    how to use a function or a function block in ST language, *30*

L

LogRecord
    Function Block, *16*

N

NumberOfBufferedRecords
    Variables, *20*
NumberOfRecords
    Variables, *20*

O

Output Parameter Type
    ERROR, *17*, *18*

T

Type
    FILE_STATUS, *20*

V

Variables
    DumpInProgress, *20*
    FileStatus, *20*
    NumberOfBufferedRecords, *20*
    NumberOfRecords, *20*

EcoStruxure Machine Expert
FileFormatUtility
Library Guide

12/2019

EIO0000002785.01

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

Safety Information . . . 7
About the Book . . . 11
Part I General Information . . . 17
Chapter 1 Presentation of the Library . . . 19
General Information . . . 19
Chapter 2 Common Inputs and Outputs . . . 21
Behavior of Function Blocks with the Input `i_xExecute` . . . 21
Part II Data Unit Types . . . 23
Chapter 3 Enumerations . . . 25
`ET_XmlItemType` . . . 26
`ET_CsvReadMode` . . . 27
`ET_ModeFileOpen` . . . 28
`ET_Result` . . . 29
Chapter 4 Structures . . . 33
`ST_XmlItem` . . . 34
`ST_XmlUserDefinedHeader` . . . 36
`ST_CsvTable` . . . 37
`ST_CsvFileInformation` . . . 39
`ST_CsvWarnValueTruncated` . . . 40
`ST_CsvReadParameter` . . . 41
`ST_CsvWriteParameter` . . . 42
Chapter 5 Aliases . . . 43
Aliases . . . 43
Part III Global Variables . . . 45
Chapter 6 Global Constants List . . . 47
Global Constants List (GCL) . . . 47
Chapter 7 Global Parameter List . . . 49
GPL . . . 49
Chapter 8 Global Variable List . . . 51
Global Variable List (GVL) . . . 51
Part IV Global Functions . . . 53
Chapter 9 Global Functions . . . 55
`FC_EtResultToString` . . . 55

Part V Generic File Handling 57
Chapter 10 Generic File Handling 59
10.1 FB_WriteFile 60
FB_WriteFile Functional Description 61
FB_WriteFile Considerations 63
FB_WriteFile Troubleshooting 63
Part VI XML Program Organization Units (POU) 65
Chapter 11 XML Function Blocks 67
11.1 FB_XmlRead 68
FB_XmlRead Functional Description 69
FB_XmlRead Considerations 72
FB_XmlRead Troubleshooting 73
FB_XmlRead Example 74
11.2 FB_XmlWrite 76
FB_XmlWrite Functional Description 77
FB_XmlWrite Considerations 80
FB_XmlWrite Troubleshooting 81
FB_XmlWrite Example 82
11.3 FB_XmlItemsUtility 84
FB_XmlItemsUtility Functional Description 85
SelectElement (Method) 86
ClearSelection (Method) 88
SelectFirstChild (Method) 89
SelectNext (Method) 90
SelectParent (Method) 91
GetAttributeByName (Method) 92
SetAttributeByName (Method) 93
AddAttribute (Method) 94
RemoveAttribute (Method) 95
InitializeXmlItems (Method) 95
AddSubElement (Method) 96
AppendElement (Method) 97
Properties of FB_XmlItemsUtility 99
Chapter 12 XML Functions 101
12.1 FC_XmlGetElementValue 102
FC_XmlGetElementValue Functional Description 103
FC_XmlGetElementValue Considerations 104

12.2 FC_XmlSetElementValue ... 105
FC_XmlSetElementValue Functional Description ... 106
FC_XmlSetElementValue Considerations ... 107
Part VII CSV Program Organization Units (POU) ... 109
Chapter 13 CSV Function Blocks ... 111
13.1 FB_CsvRead ... 112
FB_CsvRead Functional Description ... 113
FB_CsvRead Considerations ... 116
FB_CsvRead Troubleshooting ... 117
FB_CsvRead Example ... 118
13.2 FB_CsvWrite ... 120
FB_CsvWrite Functional Description ... 121
FB_CsvWrite Considerations ... 123
FB_CsvWrite Troubleshooting ... 124
FB_CsvWrite Example ... 125
Part VIII JSON Program Organization Units (POU) ... 127
Chapter 14 JSON Function Blocks ... 129
14.1 FB_CreateJsonFormattedString ... 130
FB_CreateJsonFormattedString Functional Description ... 130
New (Method) ... 131
Copy (Method) ... 131
ObjectOpen (Method) ... 132
ObjectClose (Method) ... 133
AddNameValuePairObject (Method) ... 134
AddNameValuePairArray (Method) ... 135
AddNameValuePair (Method) ... 136
AddNameValuePair<data type> (Method) ... 137
AddNameValuePairNull (Method) ... 139
ArrayAddArray (Method) ... 140
ArrayAddValue (Method) ... 140
ArrayAdd<data type> (Method) ... 142
ArrayAddNull (Method) ... 143
ArrayClose (Method) ... 144
Properties of FB_CreateJsonFormattedString ... 145
FB_CreateJsonFormattedString Example ... 146
Index ... 151

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge and experience in industrial control programming.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

BEFORE YOU BEGIN

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

⚠ WARNING

UNGUARDED EQUIPMENT

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

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

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

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

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

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

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

START-UP AND TEST

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

⚠ WARNING

EQUIPMENT OPERATION HAZARD

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

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

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

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

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

Before energizing equipment:

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

OPERATION AND ADJUSTMENTS

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

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

About the Book

At a Glance

Document Scope

This document describes the library FileFormatUtility.

The FileFormatUtility library implements functions which provide simplified access to files of certain formats.

The following file formats are supported:
- XML (eXtensible Markup Language)
- CSV (Comma Separated Values)
- JSON (JavaScript Object Notation)

Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert V1.2.

The technical characteristics of the devices described in the present document also appear online. To access the information online:

Step	Action
1	Go to the Schneider Electric home page *www.schneider-electric.com*.
2	In the **Search** box type the reference of a product or the name of a product range. ● Do not include blank spaces in the reference or product range. ● To get information on grouping similar modules, use asterisks (*).
3	If you entered a reference, go to the **Product Datasheets** search results and click on the reference that interests you. If you entered the name of a product range, go to the **Product Ranges** search results and click on the product range that interests you.
4	If more than one reference appears in the **Products** search results, click on the reference that interests you.
5	Depending on the size of your screen, you may need to scroll down to see the datasheet.
6	To save or print a datasheet as a .pdf file, click **Download XXX product datasheet**.

The characteristics that are presented in the present document should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the document and online information, use the online information as your reference.

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.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

⚠ 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.

Incomplete file transfers, such as data files, application files and/or firmware files, may have serious consequences for your machine or controller. If you remove power, or if there is a power outage or communication interruption during a file transfer, your machine may become inoperative, or your application may attempt to operate on a corrupted data file. If an interruption occurs, reattempt the transfer. Be sure to include in your risk analysis the impact of corrupted data files.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION, DATA LOSS, OR FILE CORRUPTION

- Do not interrupt an ongoing data transfer.
- If the transfer is interrupted for any reason, re-initiate the transfer.
- Do not place your machine into service until the file transfer has completed successfully, unless you have accounted for corrupted files in your risk analysis and have taken appropriate steps to prevent any potentially serious consequences due to unsuccessful file transfers.

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

Care must be taken and provisions made for use of this library for machine control to avoid inadvertent consequences of commanded machine operation, state changes, or alteration of data memory or machine operating elements.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Place operator devices of the control system near the machine or in a place where you have full view of the machine.
- Protect operator commands against unauthorized access.
- If remote control is a necessary design aspect of the application, ensure that there is a local, competent, and qualified observer present when operating from a remote location.
- Configure and install the Run/Stop input, if so equipped, or, other external means within the application, so that local control over the starting or stopping of the device can be maintained regardless of the remote commands sent to it.

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

Using the function blocks for modifying or creating files implies that you access the file system of the controller and by this also files created and / or used by the system or other processes.

NOTICE
DATA LOSS OR FILE CORRUPTION ● Verify the file path provided to the function block for file modifications. ● Verify the selected mode for opening a file to help to prevent deleting a file unintentionally. ● Create a backup of the file you intend to modify. **Failure to follow these instructions can result in equipment damage.**

Related Documents

Document title	Reference
EcoStruxure Machine Expert Functions and Libraries User Guide	*EIO0000002829 (ENG)*; *EIO0000002830 (FRE)*; *EIO0000002831 (GER)*; *EIO0000002832 (ITA)*; *EIO0000002833 (SPA)*; *EIO0000002834 (CHS)*;
EcoStruxure Machine Expert Programming Guide	*EIO0000002854 (ENG)*; *EIO0000002855 (FRE)*; *EIO0000002856 (GER)*; *EIO0000002857 (ITA)*; *EIO0000002858 (SPA)*; *EIO0000002859 (CHS)*;

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.

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

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

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

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

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

Part I
General Information

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
1	Presentation of the Library	19
2	Common Inputs and Outputs	21

Chapter 1
Presentation of the Library

General Information

Library Overview

The FileFormatUtility library implements functions which provide simplified access to the files on the file system of the controller. Further, it provides a set of functions to facilitate creating a text STRING in JavaScript Object Notation (JSON) format.

For file access, the following functions are supported:

- Simplified read and write access to files in XML (eXtensible Markup Language) and CSV (Comma-Separated Values) format.
- General write access to a file on the file system of the controller, regardless of the format.

Characteristics of the Library

The table indicates the characteristics of the library:

Characteristic	Value
Library title	FileFormatUtility
Company	Schneider Electric
Category	• **Util** • **Application/Util**
Component	**FileFormatUtility**
Default namespace	**FFU**
Language model attribute	qualified-access-only *(see EcoStruxure Machine Expert, Functions and Libraries User Guide)*
Forward compatible library	Yes (FCL *(see EcoStruxure Machine Expert, Functions and Libraries User Guide)*)

NOTE: For this library, qualified-access-only is set. This means, that the POUs, data structures, enumerations, and constants have to be accessed using the namespace of the library. The default namespace of the library is **FFU**.

Example Project

In conjunction with the library, the example projects CSV File Handling and XML File Handling are provided. The example projects demonstrate how to implement the components from the FileFormatUtility library.

The example project is installed on your PC along with the programming software. To open the project example, proceed as follows:

Step	Action	Comment
1	In the EcoStruxure Machine Expert Logic Builder, execute the command **New Project**.	–
2	In the **New Project** dialog box, select **From Example** from the **Project type** list.	–
3	On the right-hand side of the **New Project** dialog box, select the controller (1) from the **Controller** list.	**Result**: Available examples are listed in the **Matching Examples** text box.
4	Select your example from the **Matching Examples** list.	–
5	Enter a name for the new project, and select the file location.	–
6	Click the **OK** button.	**Result**: A new project is created based on the selected example.

Chapter 2
Common Inputs and Outputs

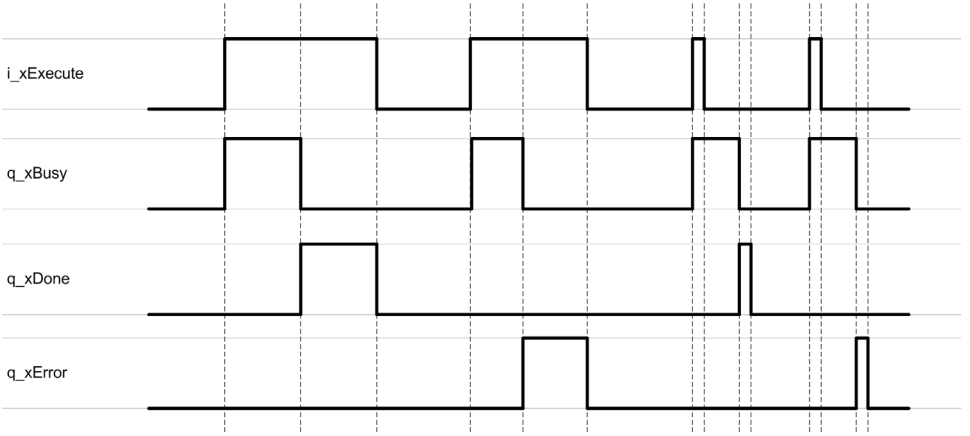
Behavior of Function Blocks with the Input `i_xExecute`

General Information

A rising edge of the input `i_xExecute` starts the execution of the function block. The function block continues execution and the output `q_xBusy` is set to TRUE. Additional rising edges at the input `i_xExecute` are ignored while the function block is being executed.

Once the execution is finished, the outputs `q_xDone` or `q_xError` remain TRUE until the input `i_xExecute` is set to FALSE. If the input is reset before the execution is finished, the outputs `q_xDone` or `q_xError` are set to TRUE for one cycle.

Example



Part II
Data Unit Types

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
3	Enumerations	25
4	Structures	33
5	Aliases	43

Chapter 3
Enumerations

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
ET_XmlItemType	26
ET_CsvReadMode	27
ET_ModeFileOpen	28
ET_Result	29

ET_XmlItemType

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration ET_XmlItemType specifies the type of an XML item.

Enumeration Elements

Name	Value (INT)	Description
NotSet	0	No type has been specified for the XML item.
Element	1	The item is of type element.
Attribute	2	The item is of type attribute.

Used By

- ST_XmlItem

ET_CsvReadMode

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration ET_CsvReadMode defines the content to be read from the CSV file using the function block FB_CsvRead.

Enumeration Elements

Name	Value (INT)	Description
AllValues	0	All values are read from the CSV file.
OneRow	1	One row (record) is read from the CSV file.
OneColumn	2	One column is read from the CSV file.
OneValue	3	One single value is read from the CSV file.
GetFileInformation	4	Only the information about the content of the file is retrieved. No value is read.

Used By

- FB_CsvRead

ET_ModeFileOpen

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration ET_ModeFileOpen specifies the mode for opening a file.

Enumeration Elements

Name	Value (INT)	Description
NotSet	0	No mode has been selected.
Append	1	An existing file is opened and the specified content is appended. If the file does not exist, the function block indicates an error.
AppendPlus	2	As for Append, an existing file is opened and the specified content is appended but with AppendPlus, if the file does not exist, a new file is created.
Create	3	A file is created and the specified content is written. If the file already exists, the function block indicates an error.
CreatePlus	4	As for Create, a file is created and the specified content is written but with CreatePlus, if the file already exists, the content is overwritten.

Used By

- FB_CsvWrite

ET_Result

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration `ET_Result` contains the possible values that indicate the result of operations executed by the POUs of this library.

Enumeration Elements

Name	Value (UDINT)	Description
`Idle`	0	The function or function block is ready to be executed.
Status information, indicated by `q_etResult` if `q_xDone` = TRUE		
`OK`	1	The function block or method has been executed successfully.
Status information, indicated by `q_etResult` if `q_xError` = FALSE and `q_xBusy` = TRUE		
`CheckingInputs`	10	The inputs are being verified.
`Initializing`	15	An internal resource is being initialized.
`OpeningFile`	17	The file is being opened.
`AnalyzingFile`	20	The file is being analyzed.
`ReadingFile`	24	The file is being read.
`WritingFile`	25	The file is being written.
`ClosingFile`	30	The file is being closed.
`GetFileSize`	35	The size of the file is being retrieved.
Error information, indicated by `q_etResult` if `q_xError` = TRUE		
`FilePathInvalid`	100	The specified file path has invalid syntax.
`XPathExpressionInvalid`	102	The specified XPath (XML Path language) expression has an invalid syntax or is not supported.
`FileInvalid`	103	The content of the specified file to be read is not supported.
`Timeout`	104	A timeout has expired during execution.
`FileOpenFailed`	110	An error has been detected while opening the file.
`FileWriteFailed`	111	An error has been detected while writing the file.
`FileCloseFailed`	112	An error has been detected while closing the file.

Name	Value (UDINT)	Description
FileAlreadyExists	113	The specified file for the write operation already exists. It is not allowed to be overwritten.
FileNotExists	114	The specified file for the write operation does not exist. It is not possible to append data.
GetFileSizeFailed	115	An error has been detected while retrieving the size of the file.
NumOfParentsExceeded	120	The nesting depth of the XML structure is greater than specified by the parameter Gc_udiXmlMaxNumOfParents in the global parameter list *(see page 49)*.
BufferFull	140	The size of the buffer for storing the read items is insufficient.
AdditionalContentInvalid	150	The pointer pbyAdditionalContent *(see page 36)* is 0 while the value of udiNumBytesToWrite is > 0.
ElementNotFound	160	No element has been found that matches the XPath expression.
XmlStructureInconsistent	165	The parent-child relations between the elements in the array XmlItems are not consistent.
XmlItemTypeInvalid	167	The type of an item in the array XmlItems is invalid.
ParsingFailed	169	While parsing the file, an internal error has been detected.
FileInconsistent	170	The structure of the parsed XML file is inconsistent. It contains at least one tag which is not properly closed.
TableReadValuesInvalid	171	The dimensions provided for the table where the read values shall be stored are not valid. Refer to ST_CsvTable *(see page 37)*.
TableWriteValuesInvalid	172	The dimensions provided for the table which contains the values that shall be written are not valid. Refer to ST_CsvTable *(see page 37)*.
TableInvalid	175	The pointer pbyAdditionalContent *(see page 36)* to the buffer provided by the application must not be 0.
ReadParameterInvalid	181	The parameters to control the read operation are not valid. Refer to ST_CsvReadParameter *(see page 41)*.
WriteParameterInvalid	182	The parameters to control write operation are not valid. Refer to ST_CsvWriteParameter *(see page 42)*.
FileReadFailed	190	While reading the file, an internal error has been detected.
TableTooSmall	200	The table does not provide enough cells to store the number of values which shall be read from the file.
ValueNotFound	210	The value specified by the row and column does not exist in the CSV file.

Name	Value (UDINT)	Description
FilePathTooLong	215	The specified file path without file extension characters is outside the valid range. It is not possible to add the default file extension. The file path without file extension is limited to 255 characters minus length of the default file extension.
FilenameTooLong	216	The specified file name including the file extension exceeds the allowed length of 126 characters.
FilenameInvalid	217	The specified file name is invalid.
FirstItemNoElement	220	The first item in the array XmlItems *(see page 34)* is not of type element.
FirstItemInvalidParentIndex	221	The value for diParentIndex of the first item in the array XmlItems *(see page 34)* is not -1.
XpathRootElementDoesNotMatch	222	The name of the first element in the array XmlItems *(see page 34)* does not match the root element specified in the XPath expression.
XpathExpressionNotSupported	230	The specified XPath expression is not supported by this function.
NoElementSelected	240	No element selected for processing the XML data.
InvalidInput	241	The value for at least one input of the executed function is invalid.
AttributeNotFound	242	Attribute of the selected element is not found.
OnlineChangeDetected	243	The execution was aborted due to a login in conjunction with an online modification of the application was detected.
WrongLayerToAppendElement	244	AppendElement() is not allowed, because the selected element is the root element.
AttributeAlreadyExistsForSelectedElement	245	Attribute of the selected element already exists.
DataTypeNotSupported	300	The data type of the variable assigned to the input of type ANY is not supported.
NoObjectOpen	301	There is no open object in the STRING that is being processed.
NoArrayOpen	302	There is no open ARRAY in the STRING that is being processed.
NameValuePairCannotBeAdded	303	A name/value pair cannot be added at the present position in the STRING that is being processed.

Name	Value (UDINT)	Description
MaxLengthOfJsonStringReached	304	The maximum length of the JSON-formatted STRING that is being processed has been reached. The maximum length is specified by the parameter Gc_udiJsonMaxLengthOfString in the global parameter list *(see page 49)*.
BufferTooSmall	305	The size of the buffer is insufficient for copying the STRING that is being processed.
NumOfLevelsExceeded	306	The maximum number of levels is reached for the JSON-formatted STRING that is being processed. The maximum number of levels is specified by the parameter Gc_uiJsonMaxNumOfLevels in the global parameter list *(see page 49)*.
BufferInvalid	400	The pointer to the buffer is not valid.
BufferSizeInvalid	401	The size of the buffer must not be 0.
UnexpectedProgramBehaviour	999	An internal error has been detected. Contact your Schneider Electric support.

Used By

- FB_XmlRead
- FB_XmlWrite
- FB_CsvRead
- FB_CsvWrite
- FB_CreateJsonFormattedString
- FB_WriteFile
- FB_XmlItemsUtility

Chapter 4
Structures

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
ST_XmlItem	34
ST_XmlUserDefinedHeader	36
ST_CsvTable	37
ST_CsvFileInformation	39
ST_CsvWarnValueTruncated	40
ST_CsvReadParameter	41
ST_CsvWriteParameter	42

ST_XmlItem

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_XmlItem is used to describe an element or an attribute that is read from or written to an XML file.

Structure Elements

Name	Data type	Description
diParentIndex	DINT	Indicates the index of the array where the parent element of the XML item is located (refer to *Example for Hierarchical Relations Indicated by uiParentIndex (see page 35)*). **NOTE:** The value of -1 indicates the root element and is reserved for the very first item of the array.
sName	STRING[GPL.Gc_uiXmlLengthOfString]	Name of the element or attribute.
sValue	STRING[GPL.Gc_uiXmlLengthOfString]	Value of the element or attribute.
etType	ET_XmlItemType *(see page 26)*	Indicates the type of the XML item.
uiNumOfAttributes	ULINT	This value depends on the type of the XML item: ● If the item is of type element, then the value indicates the number of associated attributes. ● If the item is of type attribute, then the value indicates the sequential number.

Used By

- XmlItems

Example for Hierarchical Relations Indicated by `uiParentIndex`

The example illustrates the correlation between the parameter `uiParentIndex` in the buffer of type `XmlItems` provided by the application and the hierarchical structure in the XML document.

XML document

```
<?xml version="1.0" encoding="ASCII"?>
<AAA>
   <BBB />
   <CCC />
     <DDD>
       <EEE />
     </DDD>
</AAA>
```

Buffer of type `XmlItems` provided by the application

Array Index	uiParentIndex	sName	..
0	-1	AAA	
1	0	BBB	
2	0	CCC	
3	0	DDD	
4	3	EEE	

Element	Parent index	Explanation
AAA	–1	AAA is the root element. It has no parent.
BBB	0	AAA is the parent element. It is stored in array index 0.
CCC	0	
DDD	0	
EEE	3	DDD is the parent element. It is stored in array index 3.

ST_XmlUserDefinedHeader

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_XmlUserDefinedHeader allows you to define a header that is written to the XML file.

Structure Elements

Name	Data type	Description
sUserComment	STRING[255]	Enter a text that is converted to an XML comment. It is written at the beginning of the XML file.
pbyAdditionalContent	POINTER TO BYTE	Pointer to the buffer provided by the application. It contains the content that is to be written at the beginning of the XML file in addition to the comment.
udiNumBytesToWrite	UDINT	Specifies the size in bytes for the additional content.

Used By

- FB_XmlWrite

ST_CsvTable

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure `ST_CsvTable` is used to pass the buffer provided by the application to the corresponding function block.

Structure Elements

Name	Data type	Description
`pbyTable`	POINTER TO BYTE	Pointer to the buffer (two-dimensional ARRAY of type STRING) provided by the application.
`uiNumOfRows`	UINT	Specifies the number of rows (records) in the table.
`uiNumOfColumns`	UINT	Specifies the number of values per row (record) in the table.
`udiSizeOfTable`	UDINT	Specifies the total size in bytes of the table.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator *(see EcoStruxure Machine Expert, Programming Guide)* `SIZEOF` in conjunction with the targeted buffer to determine the value for `udiSizeOfTable`.

Example

The example shows how to assign the values to this structure:

```
PROGRAM POU
VAR
    g_asCsvTable:ARRAY[0..c_uiNumOfRows-1,0..c_uiNumOfColumns-1] OF
STRING(c_uiLengthOfValue);
        stCsvTable: FFU.ST_CsvTable;
END_VAR
VAR CONSTANT
        c_uiNumOfRows :UINT:= 100;
        c_uiNumOfColumns :UINT:= 10;
        c_uiLengthOfValue :UINT:= 40;
END_VAR
stCsvTable.pbyTable := ADR(g_asCsvTable);
stCsvTable.uiNumOfRows := c_uiNumOfRows;
stCsvTable.uiNumOfColumns := c_uiNumOfColumns;
stCsvTable.udiSizeOfTable := SIZEOF(g_asCsvTable);
```

Used By

- FB_CsvRead
- FB_CsvWrite

ST_CsvFileInformation

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_CsvFileInformation provides information about the CSV file that has recently been processed by the function block FB_CsvRead *(see page 112)*.

Structure Elements

Name	Data type	Description
udiFileSize	UDINT	Indicates the size of the CSV file in bytes.
udiNumOfValues	UDINT	Indicates the number of values in the CSV file.
udiNumOfRows	UDINT	Indicates the number of rows (records) in the CSV file.
udiNumOfColumns	UDINT	Indicates the number of columns (values per record) in the CSV file.
xTableInconsistent	BOOL	Indicates TRUE if a different number of columns has been detected for at least for two rows.

Used By

- FB_CsvRead

ST_CsvWarnValueTruncated

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_CsvWarnValueTruncated provides information about the first value that has been truncated during the execution of the function block FB_CsvRead.

Structure Elements

Name	Data type	Description
xValueTruncated	BOOL	Indicates TRUE if at least one value has been truncated.
udiRow	UDINT	Indicates the number of the row in which the first value has been truncated.
udiColumn	UDINT	Indicates the number of the column in which the first value has been truncated.

Used By

- FB_CsvRead

ST_CsvReadParameter

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_CsvReadParameter is used to specify the content to be read from the CSV file using the function block FB_CsvRead.

Structure Elements

Name	Data type	Description
sDelimiter	STRING[5]	Specifies the character code for the separator that is used as delimiter between two values.
etReadMode	ET_CSVReadMode	Specifies the content to be read from the CSV file. Refer to ET_CSVReadMode *(see page 27)*.
udiNumOfRow	UDINT	Specifies the number of the row to be read. This value is relevant for: ● ET_ReadMode.OneRow ● ET_ReadMode.OneValue Refer to ET_CSVReadMode *(see page 27)*.
udiNumOfColumn	UDINT	Specifies the number of the column to be read. This value is relevant for: ● ET_ReadMode.OneColumn ● ET_ReadMode.OneValue Refer to ET_CSVReadMode *(see page 27)*.

Used By

- FB_CsvRead

ST_CsvWriteParameter

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_CsvWriteParameter is used to configure the write operation that is executed by the function block FB_CsvWrite.

Structure Elements

Name	Data type	Description
sDelimiter	STRING[5]	Specifies the character code for the separator that is inserted between two values.
etModeFileOpen	ET_ModeFileOpen	Specifies the write mode for opening or creating the CSV file. Refer to ET_ModeFileOpen *(see page 28)*.
udiNumOfRow	UDINT	Specifies the number of rows that shall be written. If this value is 0, the rows specified by the parameter i_stBufferWriteValues.uiNumOfRows will be written to the file.
udiNumOfColumn	UDINT	Specifies the number of columns per row that shall be written. If this value is 0, the columns specified by the parameter i_stBufferWriteValues.uiNumOfColumns will be written to the file.

Used By

- FB_CsvWrite

Chapter 5
Aliases

Aliases

Overview

Type:	ALIAS (DUT)
Available as of:	V1.0.8.0
Inherits from:	–

Description

An alias represents a complex data type which is used in this library.

XmlItems

Name	Data type	Description
XmlItems	ARRAY[0..GPL.Gc_udiXmlMaxNbOfElements] OFST_XmlElement	The device unit type XmlItems is used to store the elements that are read from or written to an XML file. Use it to declare the buffer provided by the application. It is linked to the function blocks FB_XmlRead and FB_XmlWrite.

Part III
Global Variables

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
6	Global Constants List	47
7	Global Parameter List	49
8	Global Variable List	51

Chapter 6
Global Constants List

Global Constants List (GCL)

Overview

Type:	Global constants
Available as of:	V1.0.8.0

Description

The global constants list contains the global constants of the FileFormatUtility library.

Global Constants

Variable	Data type	Value	Description
`Gc_sLibraryVersion`	STRING[80]	Vx.x.x.0[1]	Library version
[1] This value varies to indicate the version of the library.			

Chapter 7
Global Parameter List

GPL

Overview

Type:	Global parameters
Available as of:	V1.0.8.0

Description

The global parameter list (GPL) contains global constants which are used by certain components of this library. The parameters can be edited individually for each application where the library is used. The modification must be done within the **Library Manager** of the project where the library is referenced.

Global Parameters

Variable	Data type	Default value	Range	Description
`Gc_udiXmlMaxNumOfItems`	UDINT	1000	1... 2147483647	Determines the size of the buffer which contains the elements with their attributes which were read from or will be written to an XML file. The value indicates the sum of the elements and attributes which can be stored in the buffer.
`Gc_uiXmlLengthOfString`	UINT	40	1...254	Determines the maximum length of the elements of type STRING in the `ST_XmlItem` structure *(see page 34)*.
`Gc_udiXmlMaxNumOfParents`	UDINT	20	1...10000	Determines the maximum nesting depth of the XML structure. The variable is used internally to determine the parent index.

Variable	Data type	Default value	Range	Description
Gc_uiXmlWriteProcessingBlockSize	UINT	5000	500...65535	Determines the size of the temporary buffer (in bytes) that is used for processing the content of the XML file during write operation.
Gc_uiCsvReadProcessingBlockSize	UINT	1000	100...65535	Determines the size of the temporary buffer (in bytes) that is used for processing the values of the CSV file during read operation.
Gc_uiCsvWriteProcessingBlockSize	UINT	1000	100...65535	Determines the size of the temporary buffer (in bytes) that is used for processing the values of the CSV file during write operation.
Gc_uiJsonMaxLengthOfName	UINT	80	20...32766	Determines the maximum length of a `name` tag of a name/value pair in a JSON-formatted STRING.
Gc_uiJsonMaxLengthOfValue	UINT	80	20...32766	Determines the maximum length of a `value` tag of a name/value pair in a JSON-formatted STRING.
Gc_udiJsonMaxLengthOfString	UDINT	10000	255...32766	Determines the maximum length of the whole JSON-formatted STRING.
Gc_udiJsonMaxNumOfLevels	UINT	100	10...1000	Determines the maximum depth of the JSON-formatted STRING in which objects and ARRAYS can be nested in each other.

Chapter 8
Global Variable List

Global Variable List (GVL)

Overview

Type:	Global variables
Available as of:	V1.0.0.0

Description

The global variables list contains the global variables of the library.

Global Variables

Variable	Data Type	Default value	Description
`Gc_xOnlineChangeAllowed`	BOOL	FALSE	If TRUE, the online change detection is successfully initialized. Execute an online change while the POU is busy only if `Gc_xOnlineChangeAllowed` = TRUE.

Part IV
Global Functions

Chapter 9
Global Functions

FC_EtResultToString

Overview

Type:	Function
Available as of:	V1.0.8.0
Inherits from:	–
Implements:	–



Task

Convert an enumeration element of type ET_Result to a variable of type STRING.

Functional Description

Using the function `FC_EtResultToString`, you can convert an enumeration element of type ET_Result to a variable of type STRING.

Interface

Input	Data type	Description
`i_etResult`	ET_Result	Enumeration with the result.

Return Value

Data type	Description
STRING(80)	The ET_Result converted to text. If `i_etResult` is indeterminable the return value is: `Unknown Result: <Value of the input i_etResult>`

Part V
Generic File Handling

Chapter 10
Generic File Handling

Section 10.1
FB_WriteFile

What Is in This Section?

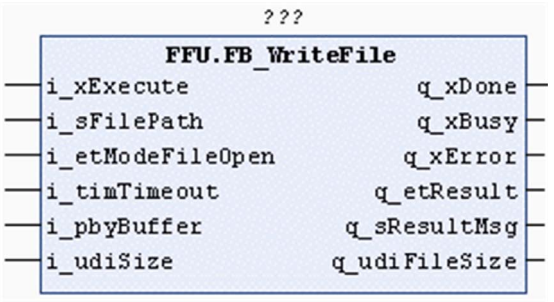
This section contains the following topics:

Topic	Page
FB_WriteFile Functional Description	61
FB_WriteFile Considerations	63
FB_WriteFile Troubleshooting	63

FB_WriteFile Functional Description

Overview

Type:	Function block
Available as of:	V1.2.0.3
Inherits from:	-
Implements:	-



Functional Description

The function block FB_WriteFile is used to open or to create a file on the file system of the controller, or on the extended memory (for example, an SD memory card), and to write specified content into it. For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller. The file format and the content to be written have no influence on the function block.

The value of the ET_ModeFileOpen enumeration *(see page 28)* allows you to specify whether the data is to be appended to an existing file or whether a new file is to be created.

The content to be written into the file is provided by a pointer to the buffer in the application memory where the data is stored. The size of data is specified in bytes and must not exceed the size of the buffer.

Interface

Input	Data type	Description
i_xExecute	BOOL	The function block opens or creates the specified file and writes the specified content upon a rising edge of this input. Also refer to the chapter *Behavior of Function Blocks with the Input i_xExecute* *(see page 21)*.

Input	Data type	Description
i_sFilePath	STRING[255]	Path to the file.
i_etModeFileOpen	ET_ModeFileOpen	Specifies the mode for opening the file and the content to be written into the file.
i_timTimeout	TIME	After this time has elapsed, the execution is canceled. If the value is T#0s, the default value T#2s is applied.
i_pbyBuffer	POINTER TO BYTE	Pointer to the buffer provided by the application. It contains the content that is to be written to the file.
i_udiSize	UDINT	Specifies the number of bytes to write. The value must not exceed the size of the buffer.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator *(see EcoStruxure Machine Expert, Programming Guide)* SIZEOF in conjunction with the targeted buffer to determine the value for i_udiSize as demonstrated in the following example.

```
fbWriteFile.i_pbyBuffer := ADR(abyBuffer);
fbWriteFile.i_udiSize := SIZEOF(abyBuffer);
```

Output	Data type	Description
q_xDone	BOOL	If this output is set to TRUE, the execution has been completed successfully.
q_xBusy	BOOL	If this output is set to TRUE, the function block execution is in progress.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to q_etResult and q_etResultMsg.
q_etResult	ET_Result	Provides diagnostic and status information as a numeric value. If q_xBusy = TRUE, the value indicates the status. If q_xDone or q_xError = TRUE, the value indicates the result.
q_sResultMsg	STRING[80]	Provides additional diagnostic and status information as a text message.
q_udiFileSize	UDINT	Provides the file size in bytes of the file recently processed.

Usage of Variables of Type POINTER TO … or REFERENCE TO …

The function block provides inputs and/or in/outputs of type POINTER TO… or REFENCE TO…. With the use of such a pointer or reference, the function block accesses the addressed memory area. In case of an online change event, it may happen that memory areas are moved to new addresses and in consequence a pointer or reference becomes invalid. To prevent errors associated with invalid pointers, variables of type POINTER TO… or REFERENCE TO… must be updated cyclically or at least at the beginning of the cycle in which they are used.

FB_WriteFile Considerations

Considerations

Consider the following constraints for opening or creating files on the file system of the controller, or on the extended memory:

- Do not change the data from the buffer provided via `i_pbBuffer` until the output `q_xBusy` of function block indicates TRUE.
- Do not access the same file with different function blocks at the same time. Use the output `q_xBusy` to lock the execution of different function blocks.

FB_WriteFile Troubleshooting

Troubleshooting

This table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified directory is not available. • The given file path has an invalid syntax. • The specified file does not exist on the specified file path. • The file extension* of specified file path is not supported.	• Verify that the directory exists. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator while another controller supports '/' instead. • For the parameter `i_etModeFileOpen` *(see page 28)*, select the value `AppendPlus`. • Verify that you do not use a file with an extension that is not supported.
Execution ends with a detected error and result indicates `FileAlreadyExists`.	• The specified file already exists but the parameter `i_etModeFileOpen` does not allow overwriting the file.	• Specify another file name. • If the existing file can be overwritten, select for the parameter `i_etModeFileOpen` *(see page 28)* the value `CreatePlus`..
Execution ends with a detected error and result indicates `Timeout`.	The specified value for the timeout is too low.	Increase the timeout parameter according to the number of cycles needed to create the file. Also consider the task interval.

* File names with file extensions .ap_, .app, .cf_, .cfg, .crc, .err, .frc, .log, .prj, .rcp,.rsi, and .urf are not supported for creating or modifying files using the `FB_WriteFile` function block.

Part VI
XML Program Organization Units (POU)

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
11	XML Function Blocks	67
12	XML Functions	101

Chapter 11
XML Function Blocks

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
11.1	FB_XmlRead	68
11.2	FB_XmlWrite	76
11.3	FB_XmlItemsUtility	84

Section 11.1
FB_XmlRead

What Is in This Section?

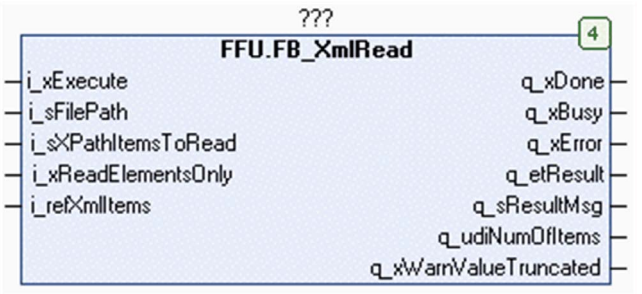
This section contains the following topics:

Topic	Page
FB_XmlRead Functional Description	69
FB_XmlRead Considerations	72
FB_XmlRead Troubleshooting	73
FB_XmlRead Example	74

FB_XmlRead Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-



Functional Description

The function block FB_XmlRead is used to read (parse) an XML file that is located on the file system of the controller, or on the extended memory (for example, an SD memory card). For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

The content of the XML file, XML elements together with their attributes and values, is stored in an array of type XmlItems in the application memory of the controller. You have to declare this array and assign it to the associated input i_refXmlItems on the function block. At the beginning of each read operation, the content of this array is erased.

When executing the function block, the input i_refXmlItems is stored internally for further use. In case an online change event is detected while the function block is executed (q_xBusy = TRUE), the internally used variables are updated with the present value of the input.

NOTE: Do not reassign the i_refXmlItems to another variable while the function block is executed.

The number of items (sum of elements and attributes) which can be stored in the array is specified by the parameter Gc_udiXmlMaxNumOfItems in the GPL *(see page 49)*.

The array contains the fields of type STRING to store the names and the values of the elements and attributes. You can specify the length of these STRINGs by the global parameter `Gc_uiXml-LengthOfString`. If a value to be read in the file exceeds the specified length, the original value is truncated. If at least one value has been truncated, this is indicated by the output `q_xWarnValueTruncated`.

NOTE: The output `q_xWarnValueTruncated` is valid only if the output `q_xDone` is TRUE.

The hierarchical structure of the elements from the XML file is indicated by the parameter `uiParentIndex` for each item in the array of type `XmlItems`. For further information, refer to `uiParentIndex` *Example for Hierarchical Relations Indicated by uiParentIndex (see page 35)*.

Interface

Input	Data type	Description
`i_xExecute`	BOOL	The function block executes the read operation with the specified XML file upon a rising edge of this input. Also refer to the chapter *Behavior of Function Blocks with the Input i_xExecute (see page 21)*.
`i_sFilePath`	STRING[255]	File path to the XML file that shall be read. If a file name is specified without file extension, the function block adds the extension .xml.
`i_sXPathItemToRead`	STRING[255]	XPath expression to address the elements which shall be read from the XML file. Default value: `'//*'`
`i_xReadElementsOnly`	BOOL	If this input is TRUE, the element names and their values are read and stored to the application buffer. If this input is FALSE, the attributes together with their values are also read and stored to the application buffer.
`i_refXmlItems`	REFERENCE TO `XmlItems`	Buffer provided by the application to store the elements read from the specified XML file. The buffer is erased with each execution of the function block.

Output	Data type	Description
`q_xDone`	BOOL	If this output is set to TRUE, the execution has been completed successfully.
`q_xBusy`	BOOL	If this output is set to TRUE, the function block execution is in progress.
`q_xError`	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to `q_etResult` and `q_etResultMsg`.
`q_etResult`	`ET_Result`	Provides diagnostic and status information as a numeric value. If `q_xBusy` = TRUE, the value indicates the status. If `q_xDone` or `q_xError` = TRUE, the value indicates the result.

Output	Data type	Description
q_sResultMsg	STRING[80]	Provides additional diagnostic and status information as a text message.
q_udiNumOfItemsRead	UDINT	Indicates the total number of elements and attributes read from the XML file.
q_xWarnValueTruncated	BOOL	If this output is set to TRUE, at least one value has been truncated. **NOTE:** The output is updated along with q_xDone.

For more information about the signal behavior of the basic inputs and outputs, refer to the chapter *Behavior of Function Blocks with the Input i_xExecute* *(see page 21)*.

Usage of Variables of Type POINTER TO … or REFERENCE TO …

The function block provides inputs and/or in/outputs of type POINTER TO… or REFENCE TO…. With the use of such a pointer or reference, the function block accesses the addressed memory area. In case of an online change event, it may happen that memory areas are moved to new addresses and in consequence a pointer or reference becomes invalid. To prevent errors associated with invalid pointers, variables of type POINTER TO… or REFERENCE TO… must be updated cyclically or at least at the beginning of the cycle in which they are used.

XPath Expressions Defining the Content to Be Read

To be able to read a single element or a group of elements from the XML file, use the syntax of the XPath (XML Path) language. The content to be read is specified by the input i_XpathItemToRead

NOTE: The function block FB_XmlRead supports a subset of the features provided with XPath expressions.

The table lists the supported XPath expressions:

XPath expression	Description
//*	Selects all elements in the document.
/	Indicates an absolute path to an element.
/…/child::*	Selects all child elements of the node.
/…/descendant::*	Selects all descendant elements of the node.
/…/<elementname>	Selects all elements with the specified name of the node.
/…/<elementname>[<n>]	Selects the nth element with the specified name of the node.
/…/<elementname>[@<attribute>]	Selects all elements with the specified name and the specified attribute of the node.
/…/<elementname>[@<attribute>=<value>]	Selects all elements with the specified name and the specified attribute and value of the node.

NOTE: The predicates, that are the expressions within square brackets `[]`, can be followed by a slash `/` together with an element name to address the next child element.

Example: `/…/<elementname>[<n>]/<elementname>`

FB_XmlRead Considerations

Considerations

Consider the following constraints for reading an XML file:

- Only XML files with ASCII encoding are supported.
- Blank spaces are interpreted as values, tabs are not interpreted as values.
- Line breaks in values are not supported. If a value includes a line break, the characters before the line break are considered during parsing.
- Only the names of elements and their attributes together with their values are read and then stored in the buffer provided by the application. This has the effect that XML objects other than elements and attributes, such as comments, and DOCTYPE declarations are not detected by the XML parser.
- CDATA objects are not supported. The content of CDATA objects is interpreted as a value of the open element.
- The values read from the file are stored as STRING values in the application. This applies even to numeric values. Before they can be processed, the values need to be converted to the appropriate datatype. To achieve this, it is a good practice to use the `STRING_TO_` conversion functions *(see EcoStruxure Machine Expert, Programming Guide)*. The conversion functions require a specific syntax of the STRING values depending on the target datatype. Consider these requirements when creating the files to simplify the processing of read values.
- Parsing the XML file is a time-consuming process. It is performed in parallel to the task that calls the function block. Thus, the time required for parsing is part of the task execution time. It results, for a single cycle, in an increased task execution time while the `FB_XmlRead` function block is executed. Depending on the size of the file and the controller, the increased task cycle can take up to a few seconds. To help prevent other processes from being blocked by parsing an XML file, create a separate task with low priority (>24) for this function. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- Do not process the data from the buffer provided via `i_refXmlItems` as long as the output `q_xBusy` of function block indicates TRUE.
- Do not access the same file with different function blocks at the same time. Use the output `q_xBusy` to lock the execution of different function blocks.

FB_XmlRead Troubleshooting

Troubleshooting

This table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified file or the directory is not available. • The given file path has invalid syntax.	• Verify that the file exists under the specified directory. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead.
Execution ends with a detected error and result indicates `FileInvalid`.	• The file contains characters outside the ASCII range. • During parsing, an unexpected closing tag has been detected.	• Verify that the file contains only ASCII characters. • Verify that the XML file is valid: Make sure that each open tag is closed and that the elements are properly nested. **NOTE:** Element names are case-sensitive.
Execution ends with a detected error and result indicates `XPathExpressionInvalid`.	• The entered XPath expression is invalid or not supported.	• Verify the correct syntax of the XPath expression. • Verify that the expression is supported by the function block *(see page 71)*.
Execution ends with a detected error and result indicates `NumOfParentsExceeded`.	• The nesting depth of elements in the XML file to read is larger than specified.	• Increase the value for the parameter `Gc_udiXmlMaxNumOfParents` in the GPL *(see page 49)*. • Avoid the use of files with deeply nested XML structures.
Execution ends with a detected error and result indicates `BufferFull`.	• The number of items in the XML file to read is greater than the size of the buffer provided by the application.	• Increase the buffer size by the parameter `Gc_XmlMaxNumOfItems` in the GPL *(see page 49)*. • Split the read operation in several blocks. Use an appropriate XPath expression to limit the number of elements to read per block.

Issue	Cause	Solution
Execution ends with a detected error and result indicates `ElementNotFound`.	• The element specified in the XPath expression does not exist in the file. • The path declaration in the XPath expression does not match the structure in the XML file. • There is a typing error in the XPath expression.	• Verify that the specified element is typed correctly. • Verify that the path declaration meets the expected structure in the XML file. • Verify that the correct XML file is specified with the parameter `i_sFilePath` *(see page 70)*.
Execution ends with a detected error and result indicates `FileInconsistent`.	• The end of file has been reached but at least one tag has not been closed.	• Verify that the XML file is valid: Make sure that each open tag is closed and that the elements are properly nested.

FB_XmlRead Example

Overview

The following example shows how the elements read from an XML file are stored in the buffer provided by the application for further processing.

Example XML File

```
<?xml version="1.0" encoding="ASCII"?>
<!--This is the user comment.-->
<!DOCTYPE AAA SYSTEM "example.dtd">

<AAA>
  <BBB>1st bbb</BBB>
  <BBB>
    <CCC id="1">
      <DDD id="1" activate="TRUE">ddd</DDD>
    </CCC>
  </BBB>
</AAA>
```

NOTE: The comment and the doctype declaration (DTD) in lines 2 and 3 are not read. This information will not be available in the application.

Example Program

```
PROGRAM SR_Example
VAR
  fbRead :FFU.FB_XmlRead;
  astXmlItems :FFU.XmlItems;
  xCmdRead :BOOL;
END_VAR
fbRead(
       i_xExecute := xCmdRead,
       i_sFilePath := '/sd0/Example.xml',
       i_sXPathItemsToRead := ,
       i_xReadElementsOnly := ,
       i_refXmlItems := astXmlItems,
       q_xDone => ,
       q_xBusy => ,
       q_xError => ,
       etResult => ,
       q_sResultMsg => ,
       q_udiNumOfItems => ,
       q_xWarnValueTruncated => ) ;
```

Buffer

The buffer provided by the application of type `XmlItems` contains the elements and attributes read from the XML file.

For this example, the value of the output `q_udiNumOfItems` is 8.

Array Index	diParentIndex	sName	sValue	etType	diNumOfAttributes
0	-1	AAA	–	1	0
1	0	BBB	1st bbb	1	0
2	0	BBB	–	1	0
3	2	CCC	–	1	1
4	3	id	1	2	1
5	3	DDD	ddd	1	2
6	5	id	1	2	1
7	5	activate	TRUE	2	2

Section 11.2
FB_XmlWrite

What Is in This Section?

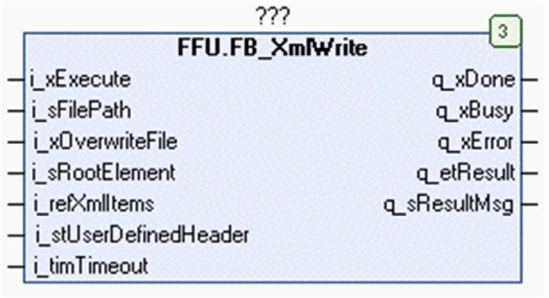
This section contains the following topics:

Topic	Page
FB_XmlWrite Functional Description	77
FB_XmlWrite Considerations	80
FB_XmlWrite Troubleshooting	81
FB_XmlWrite Example	82

FB_XmlWrite Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-



Functional Description

The function block FB_XmlWrite is used to create or overwrite an XML file on the file system of the controller, or on the extended memory (for example, an SD memory card). For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

When executing the function block, the input i_refXmlItems is stored internally for further use. In case an online change event is detected while the function block is executed (q_xBusy = TRUE), the internally used variables are updated with the present value of the input.

NOTE: Do not reassign the i_refXmlItems to another variable while the function block is executed.

After the file has been created, the elements provided in the array of type XmlItems in the application memory of the controller are written into it.

The character code LF (0A hex) is used as line break in the created file.

The prolog <?xml version="1.0" encoding="ASCII"?> is inserted as first line of each file that is created by the function block.

After the prolog, the XML elements that are provided in the array of type `XmlItems`, including their attributes and their values, are written to the file. The structure or the nesting of the elements is specified by the parameter `uiParentIndex` which is part of the structure `ST_XmlItem`. For further information, refer to `uiParentIndex` *Example for Hierarchical Relations Indicated by uiParentIndex (see page 35)*.

The function block `FB_XmlWrite` provides the input `i_sRootElement` to specify a root element. This is useful in case the array providing the elements contains no root element or more than one root element:

- The use case without root element is allowed if the values of the `uiParentIndex` parameters in the array are all 0.
- The use case with more than one root element can occur if a group of child elements has been read from a file before the right operation and the modified data is to be written to a new file.

Apart from these two exceptional cases, the defined structure of the elements must be consistent. Otherwise the function block cancels writing the file and the file is discarded.

For items of type attribute, the parameter `uiParentIndex` does not have an effect. Attributes are assigned to the next upper item of type element in the array.

You can define additional content using the structure `i_stUserDefinedHeader`. The content is written between the prolog and the first element to the XML file. This additional content could be, for example, a comment (XML syntax) and/or a DOCTYPE declaration (DTD).

The input `i_xOverwriteFile` allows you to define whether an existing file will be overwritten. If the input is FALSE and the specified file already exists, the execution of the function block is canceled and an error is indicated.

Interface

Input	Data type	Description
`i_xExecute`	BOOL	The function block creates the specified XML file and writes the specified content into it upon a rising edge of this input. Also refer to the chapter *Behavior of Function Blocks with the Input i_xExecute (see page 21)*.
`i_sFilePath`	STRING[255]	File path to the XML file. If a file name is specified without file extension, the function block adds the extension .xml.
`i_xOverwriteFile`	BOOL	Specifies whether an existing file is to be overwritten. Set this input to TRUE to allow the replacement of an existing file.
`i_sRootElement`	STRING[`GPL.Gc_uiXmlLengthOfString`]	Root element which is created if the array of structure `XmlElements` contains more than one root element.

Input	Data type	Description
i_stUserDefinedHeader	ST_XmlUserDefinedHeaderAscii	The structure contains user-defined content that is to be written at the beginning of the newly created XML file.
i_timTimeout	TIME	After this time has elapsed, the execution is canceled. If the value is T#0s, the default value T#2s is applied.
i_refXmlItems	REFERENCE TO XmlItems	Buffer provided by the application that contains the content to be written to the XML file.

Output	Data type	Description
q_xDone	BOOL	If this output is set to TRUE, the execution has been completed successfully.
q_xBusy	BOOL	If this output is set to TRUE, the function block execution is in progress.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to q_etResult and q_etResultMsg.
q_etResult	ET_Result	Provides diagnostic and status information as a numeric value. If q_xBusy = TRUE, the value indicates the status. If q_xDone or q_xError = TRUE, the value indicates the result.
q_sResultMsg	STRING[80]	Provides additional diagnostic and status information as a text message.

Usage of Variables of Type POINTER TO … or REFERENCE TO …

The function block provides inputs and/or in/outputs of type POINTER TO… or REFENCE TO…. With the use of such a pointer or reference, the function block accesses the addressed memory area. In case of an online change event, it may happen that memory areas are moved to new addresses and in consequence a pointer or reference becomes invalid. To prevent errors associated with invalid pointers, variables of type POINTER TO… or REFERENCE TO… must be updated cyclically or at least at the beginning of the cycle in which they are used.

FB_XmlWrite Considerations

Considerations

Consider the following constraints for writing an XML file:

- File operations are time-consuming processes. To help prevent influencing time-critical control functions in your application, create a separate task with lower priority for these processes. For more information about task management, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- The timeout parameter `i_timTimeout` is used to monitor the file operation. If the value specified for the timeout is exceeded during execution of the function block, the write operation is canceled and the function block indicates an error. To select an appropriate value for the timeout parameter, consider that the file operation takes several task cycles. Thus, calculate the product of the number of required task cycles and the task interval to specify the minimum value for the timeout parameter.
- The number of cycles depends, besides the amount of data to be written, on the size of the processing block which can be specified by the parameter `Gc_uiXmlWriteProcessing-BlockSize` in the GPL *(see page 49)*. The creation of the file is split into several write operations to reduce the load for the single task cycle. During each write operation, one data block is processed and written to the file. The larger the processing block size the fewer cycles are required for creating and writing the file. However, a larger block size increases the execution time of each write operation.
- Do not process the data from the buffer provided via `i_refXmlItems` as long as the output `q_xBusy` of function block indicates TRUE.
- Do not access the same file with different function blocks at the same time. Use the output `q_xBusy` to lock the execution of different function blocks.

FB_XmlWrite Troubleshooting

Troubleshooting

This table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified directory is not available. • The given file path has an invalid syntax.	• Verify that the directory exists. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead.
Execution ends with a detected error and result indicates `FileAlreadyExists`.	• The specified file already exists and the input `i_xOverwriteFile` is FALSE.	• Specify another file name. • If the existing file can be overwritten, set the input `i_xOverwriteFile` to TRUE.
Execution ends with a detected error and result indicates `XmlStructureInconsistent`.	• At least one value for the parameter `uiParentIndex` in the buffer that contains the XML elements *(see page 35)* prevents the creation of a valid XML file. • The `uiParentIndex` of the first element is <> -1.	• Verify that the values defined for the parameter `uiParentIndex` *(see page 34)* are consistent. **NOTE:** Refer to the output `q_sResultMsg` for further information on the invalid parameter.
Execution ends with a detected error and result indicates `Timeout`.	• The specified value for the timeout is too low.	• Increase the timeout parameter according to the number of cycles needed to create the file. Consider also the task interval. • Increase the processing block size per write operation to reduce the number of function block calls which are needed to create the file.

FB_XmlWrite Example

Overview

The following example shows how the elements and attributes stored in the buffer provided by the application are written to the XML file. In addition to the elements and attributes, an example for the additional content is provided.

Buffer

The buffer provided by the application of type `XmlItems` contains elements to be written to the XML file.

Array Index	diParentIndex	sName	sValue	etType	diNumOfAttributes
0	-1	AAA	–	1	0
1	0	BBB	1st bbb	1	0
2	0	BBB	–	1	0
3	2	CCC	–	1	1
4	–	id	1	2	1
5	3	DDD	ddd	1	2
6	–	id	1	2	1
7	–	activate	TRUE	2	2

Example Program

```
PROGRAM SR_Example
VAR
  fbWrite :FFU.FB_XmlWrite;
  astXmlItems :FFU.XmlItems;
  xCmdWrite :BOOL;
  sComment :STRING(255) := 'This is the user comment.';
  sExternalDTD :STRING := '<!DOCTYPE AAA SYSTEM "example.dtd">';
  stHeader :FFU.ST_XmlUserDefinedHeader;
END_VAR

stHeader.sUserComment := sComment;
stHeader.pbyAdditionalContent := ADR(sExternalDTD);
stHeader.udiNumBytesToWrite := INT_TO_UDINT(Standard.LEN(sExternalDTD))
;

fbWrite(
        i_xExecute := xCmdWrite,
        i_sFilePath := '/sd0/Example.xml',
        i_xOverwriteFile := ,
```

```
        i_sRootElement := ,
        i_refXmlItems := astXmlItems,
        i_stUserDefinedHeader := stHeader,
        i_timTimeout := ,
        q_xDone => ,
        q_xBusy => ,
        q_xError => ,
        q_etResult => ,
        q_sResultMsg => );
```

Example XML File

```
<?xml version="1.0" encoding="ASCII"?>
<!--This is the user comment.-->
<!DOCTYPE AAA SYSTEM "example.dtd">

<AAA>
  <BBB>1st bbb</BBB>
    <CCC id="1">
      <DDD id="1" activate="TRUE">ddd</DDD>
    </CCC>
  </BBB>
</AAA>
```

Section 11.3
FB_XmlItemsUtility

What Is in This Section?

This section contains the following topics:

Topic	Page
FB_XmlItemsUtility Functional Description	85
SelectElement (Method)	86
ClearSelection (Method)	88
SelectFirstChild (Method)	89
SelectNext (Method)	90
SelectParent (Method)	91
GetAttributeByName (Method)	92
SetAttributeByName (Method)	93
AddAttribute (Method)	94
RemoveAttribute (Method)	95
InitializeXmlItems (Method)	95
AddSubElement (Method)	96
AppendElement (Method)	97
Properties of FB_XmlItemsUtility	99

FB_XmlItemsUtility Functional Description

Overview

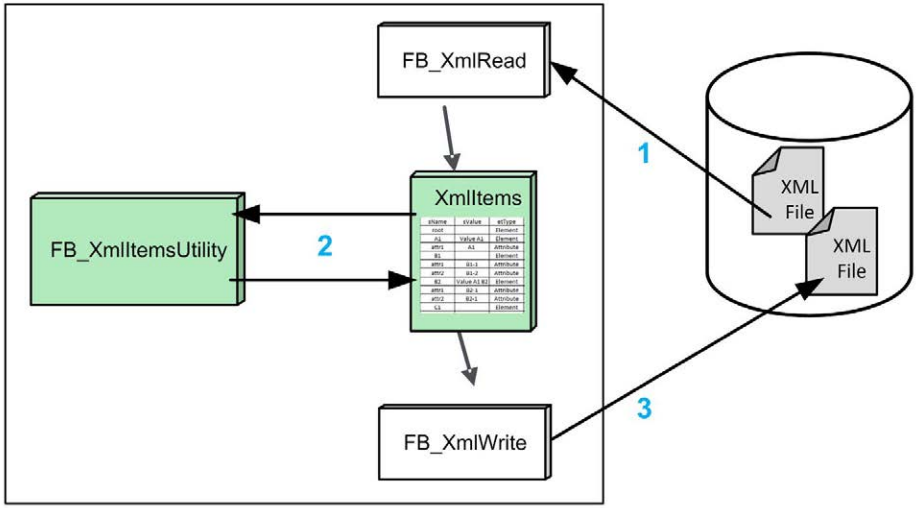
Type:	Function block
Available as of:	V1.3.2.0
Inherits from:	-
Implements:	-

Functional Description

The function block FB_XmlItemsUtility is used to process the data contained in an array of type XmlItems *(see page 34)*.

The function block provides several methods and properties for processing the data in the array.

Using the methods and properties allows you to parse the array to get or to modify the values from the single XML items. In addition, you can add further elements and attributes.



Item	Description
1	The function block `FB_XmlRead` *(see page 69)* is used to read the content from an XML file. The data is stored in an array of type `XmlItems`.
2	The function block `FB_XmlItemsUtility` supports the following features: • Reading of single values • Modifying of single values • Modifying an existing XML structure
3	The function block `FB_XmlWrite` *(see page 77)* is used to create a new XML file based on the data from the array of type `XmlItems`. An existing file can be overwritten.

Before you start processing data in the array, you must call either the method `SelectElement` *(see page 86)* or `InitializeXmlItems` *(see page 95)*. After a successful execution of one of the methods, the required link to the array is created and the desired base element for further operations is selected.

The function block does not provide parameters but methods and properties. The instance of the function block is used exclusively for holding the local variables which are accessed during the sequential processing of the associated methods or properties. Therefore, it is not required to call the function block in the application code.

`SelectElement` (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to select the specified element from the array of type `XmlItems`. Based on the selected element, further methods can be executed.

The element is specified using an XPath expression. If the XPath expression matches several elements, the first matching element is selected. If a null string is assigned to the input `i_sXpathToElement`, the root element becomes selected.

On each call, the search of the specified element is started from the beginning of the array. That is, using the same XPath expression always results in the same element being selected.

The return value of type BOOL indicates TRUE if an element was successfully selected.

A call of this method returns either `Ok`, `XPathExpressionInvalid`, or `ElementNotFound`. Use the property `Result` to obtain the result of the method.

Interface

Input	Data type	Description
`i_sXpathToElement`	STRING[255]	XPath expression to specify the element to be selected. If a null string is assigned, the root element becomes selected.
`i_refXmlItems`	REFERENCE TO `XmlItems`	Array provided by the application which contains the elements and attributes read from or to be written to an XML file.

XPath Expressions

Use the syntax of the XPath (XML Path) language to specify the element to be selected.

The table lists the supported XPath expressions:

XPath expression	Description
`/…/<elementname>`	Selects the first element matching the specified path.
`/…/<elementname>[<n>]`	Selects the nth element matching the specified path.
`/…/<elementname>[@<attribute>]`	Selects the first element matching the specified path that has the specified attribute.
`/…/<elementname>[@<attribute>=<value>]`	Selects the first element matching the specified path that has the specified attribute and value.

The predicates (the expressions within square brackets `[]`) can be followed by a slash `/` together with an element name to address the next child element.

Example: `/…/<elementname>[<n>]/<elementname>`

Example

Precondition: The XML file (illustrated below) was read using the FB_XmlRead *(see page 69)* and the content is stored in the array `astXmlFile` of type `XmlItems`.

Code:

```
fbXmlItems.SelectElement('/root/A1',
astXmlData);
```

Result:
The element A1 is selected.

```
<root>
  <A1 a1="A1-1">
    <B1 a1="B1-1" a2="B1-2"/>
    <B2 a1="B2-1"  a2="B2-2"/>
      <C1 a1="C1-1" a2="B2-2"/>
      <C2 a1="C2-1" a2="C1-2"/>
    <B3 a1="B3-1" a2="B3-2"/>
  </A1>
  <A2 a1="A2-1">
    <B1 a1="B1-1" a2="B1-2"/>
    <B2 a1="B2-1"  a2="B2-2"/>
      <C1 a1="C1-1" a2="B2-2"/>
      <C2 a1="C2-1" a2="C1-2"/>
    <B3 a1="B3-1" a2="B3-2"/>
  </A2>
</root>
```

`ClearSelection` (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to clear the present selection.

The return value of type BOOL indicates TRUE, no matter if an element was selected before or not.

A call of this method returns `0k`.

SelectFirstChild (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to select the first child element of the presently selected element.

The return value of type BOOL indicates TRUE if an element was successfully selected.

A call of this method returns either `Ok`, `NoElementSelected`, or `ElementNotFound`. Use the property `Result` to obtain the result of the method.

The method has no inputs.

Example

Precondition: The XML file (illustrated below) was read using the `FB_XmlRead` *(see page 69)* and the content is stored in the array `astXmlFile` of type `XmlItems`.

Code:

```
fbXmlItems.SelectElement('/root/A1',
astXmlData);
fbXmlItems.SelectFirstChild();
```

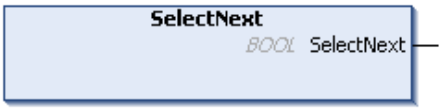
Result:
The element B1 is selected.

```
1  <root>
2    <A1 a1="A1-1">
3      <B1 a1="B1-1" a2="B1-2"/>
4      <B2 a1="B2-1"  a2="B2-2"/>
5        <C1 a1="C1-1" a2="B2-2"/>
6        <C2 a1="C2-1" a2="C1-2"/>
7      <B3 a1="B3-1" a2="B3-2"/>
8    </A1>
9    <A2 a1="A2-1">
10     <B1 a1="B1-1" a2="B1-2"/>
11     <B2 a1="B2-1"  a2="B2-2"/>
12       <C1 a1="C1-1" a2="B2-2"/>
13       <C2 a1="C2-1" a2="C1-2"/>
14     <B3 a1="B3-1" a2="B3-2"/>
15   </A2>
16 </root>
```

SelectNext (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to select the next element with the same parent as the presently selected element.

The return value of type BOOL indicates TRUE if an element was successfully selected.

A call of this method returns either `Ok`, `NoElementSelected`, or `ElementNotFound`. Use the property `Result` to obtain the result of the method.

The method has no inputs.

Example

Precondition: The XML file (illustrated below) was read using the `FB_XmlRead` *(see page 69)* and the content is stored in the array `astXmlFile` of type `XmlItems`.

Code:

```
fbXmlItems.SelectElement('/root/A1');
fbXmlItems.SelectFirstChild();
fbXmlItems.SelectNext();
fbXmlItems.SelectNext();
```

Result:
The element B3 is selected.

```
1  <root>
2    <A1 a1="A1-1">
3      <B1 a1="B1-1" a2="B1-2"/>
4      <B2 a1="B2-1"  a2="B2-2"/>
5        <C1 a1="C1-1" a2="B2-2"/>
6        <C2 a1="C2-1" a2="C1-2"/>
7      <B3 a1="B3-1" a2="B3-2"/>
8    </A1>
9    <A2 a1="A2-1">
10     <B1 a1="B1-1" a2="B1-2"/>
11     <B2 a1="B2-1"  a2="B2-2"/>
12       <C1 a1="C1-1" a2="B2-2"/>
13       <C2 a1="C2-1" a2="C1-2"/>
14     <B3 a1="B3-1" a2="B3-2"/>
15   </A2>
16 </root>
```

SelectParent (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to select the parent element of the presently selected element.

The return value of type BOOL indicates TRUE if an element was successfully selected.

A call of this method returns either `Ok`, `NoElementSelected`, or `ElementNotFound`. Use the property `Result` to obtain the result of the method.

The method has no inputs.

Example

Precondition: The XML file (illustrated below) was read using the `FB_XmlRead` *(see page 69)* and the content is stored in the array `astXmlFile` of type `XmlItems`.

Code:

```
fbXmlItems.SelectElement('/root/A1');
fbXmlItems.SelectFirstChild();
fbXmlItems.SelectNext();
fbXmlItems.SelectFirstChild();
// do something with element C1;
fbXmlItems.SelectParent();
```

Result:
The element B2 is selected.

```
1  <root>
2    <A1 a1="A1-1">
3      <B1 a1="B1-1" a2="B1-2">
4      <B2 a1="B2-1" a2="B2-2">
5        <C1 a1="C1-1" a2="C1-2">
6        <C2 a1="C2-1" a2="C2-2">
7      <B3 a1="B3-1" a2="B3-2">
8    </A1>
9    <A2 a1="A2-1">
10     <B1 a1="B1-1" a2="B1-2">
11     <B2 a1="B2-1" a2="B2-2">
12       <C1 a1="C1-1" a2="C1-2">
13       <C2 a1="C2-1" a2="C2-2">
14     <B3 a1="B3-1" a2="B3-2">
15   </A2>
16 </root>
```

GetAttributeByName (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to read the value of the specified attribute from the selected element.

The return value of type BOOL indicates TRUE if the attribute was found.

A call of this method returns either `Ok`, `NoElementSelected`, `InvalidInput`, or `AttributeNotFound`. Use the property `Result` to obtain the result of the method.

If the `q_sValue` is a null string either no element is selected, the attribute was not found, or the attribute has no value. The value on `q_sValue` is valid only if the return value of the method is TRUE.

Interface

Input	Data type	Description
`i_sAttribute`	STRING `[Gc_uiXmlLengthOfString]`	Name of the attribute.

Output	Data type	Description
`q_sValue`	STRING `[Gc_uiXmlLengthOfString]`	Value of the specified attribute.

Example

Precondition: The XML file (illustrated below) was read using the FB_XmlRead *(see page 69)* and the content is stored in the array astXmlFile of type XmlItems.

Code:

```
fbXmlItems.SelectElement('/root/A1',
astXmlData);
IF fbXmlItems.GetAttributeByName('a1',
sValue) THEN
  // do something with sValue
END_IF
```

Result:
The variable sValue provides the value 'A1-1'.

```
1  <root>
2    <A1 a1="A1-1">
3      <B1 a1="B1-1" a2="B1-2"/>
4      <B2 a1="B2-1"  a2="B2-2"/>
5        <C1 a1="C1-1" a2="B2-2"/>
6        <C2 a1="C2-1" a2="C1-2"/>
7      <B3 a1="B3-1" a2="B3-2"/>
8    </A1>
9    <A2 a1="A2-1">
10     <B1 a1="B1-1" a2="B1-2"/>
11     <B2 a1="B2-1"  a2="B2-2"/>
12       <C1 a1="C1-1" a2="B2-2"/>
13       <C2 a1="C2-1" a2="C1-2"/>
14     <B3 a1="B3-1" a2="B3-2"/>
15   </A2>
16 </root>
```

SetAttributeByName (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to set the value of the specified attribute for the selected element.

The return value of type BOOL indicates TRUE if the attribute was found and the specified value was set.

A call of this method returns either Ok, NoElementSelected, InvalidInput, or AttributeNotFound. Use the property Result to obtain the result of the method.

Interface

Input	Data type	Description
i_sAttribute	STRING [Gc_uiXmlLengthOfString]	Name of the attribute.
i_sValue	STRING [Gc_uiXmlLengthOfString]	New value of the attribute.

AddAttribute (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to add an attribute to the selected element.

The return value of type BOOL indicates TRUE if the attribute was added successfully.

A call of this method returns either `Ok`, `NoElementSelected`, `AttributeAlreadyExists-ForSelectedElement`, `InvalidInput`, or `BufferFull`. Use the property `Result` to obtain the result of the method.

Interface

Input	Data type	Description
i_sAttribute	STRING [Gc_uiXmlLengthOfString]	Name of the attribute.
i_sValue	STRING [Gc_uiXmlLengthOfString]	Value of the attribute.

RemoveAttribute (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to remove an attribute from the selected element.

The return value of type BOOL indicates TRUE if the attribute was successfully removed.

A call of this method returns either `Ok`, `NoElementSelected`, `InvalidInput`, or `AttributeNotFound`. Use the property `Result` to obtain the result of the method.

Interface

Input	Data type	Description
`i_sAttribute`	STRING `[Gc_uiXmlLengthOfString]`	Name of the attribute.

InitializeXmlItems (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to initialize the linked array of type `XmlItems`. During initialization, the content of the array is erased and the specified element is created as the root element. After the method is executed successfully, the created element is selected for further operations.

The return value of type BOOL indicates TRUE if the array was initialized successfully.

A call of this method returns either `Ok`, or `InvalidInput`. Use the property `Result` to obtain the result of the method.

Interface

Input	Data type	Description
`i_sElementName`	STRING `[Gc_uiXmlLengthOfString]`	Name of the element to be created as root.
`i_refXmlItems`	REFERENCE TO `XMLItems`	Array provided by the application which contains the elements and attributes read from or to be written to an XML file.

Example

Code:

```
fbXmlItems.InitializeXmlItems('root', astXmlData);
```

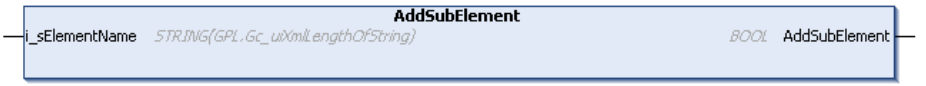
Result:
The resulting XML file only contains the element 'root'.

```
1 <root>
2 </root>
```

`AddSubElement` (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to add a new sub-element under the presently selected element. After the method is executed successfully, the new element is selected for further operations.

The return value of type BOOL indicates TRUE if a new element was added successfully.

A call of this method returns either `Ok`, `InvalidInput`, `NoElementSelected`, `MaxNumOfParentsExceeded`, or `BufferFull`. Use the property `Result` to obtain the result of the method.

Interface

Input	Data type	Description
`i_sElementName`	STRING `[Gc_uiXmlLengthOfString]`	Name of the sub-element to be added.

Example

Code:
`fbXmlItems.InitializeXmlItems('root', astXmlData);`
`fbXmlItems.AddSubElement('A1');`
`fbXmlItems.AddAttribute('a1', 'A1-1');`
Result:
The resulting XML file is displayed on the right-hand side.
	1 <root>
	2 <A1 a1="A1-1" />
	3 </root>

`AppendElement` (Method)

Overview

Type:	Method
Available as of:	V1.3.2.0



Functional Description

This method is used to append a new element at the same level of the presently selected element. After the method was executed successfully, the new element is selected for further operations.

The return value of type BOOL indicates TRUE if a new element was appended successfully.

A call of this method returns either `Ok`, `WrongLayerToAppendElement`, `InvalidInput`, `NoElementSelected`, or `BufferFull`. Use the property `Result` to obtain the result of the method.

Interface

Input	Data type	Description
`i_sElementName`	STRING `[Gc_uiXmlLengthOfString]`	Name of the element to be appended.

Example

Code:

```
fbXmlItems.InitializeXmlItems('root', astXmlData);
fbXmlItems.AddSubElement('A1');
fbXmlItems.AddAttribute('a1', 'A1-1');
fbXmlItems.AppendElement('A2');
```

Result:
The resulting XML file is displayed on the right-hand side.

```
1  <root>
2    <A1 a1="A1-1" />
3    <A2 />
4  </root>
```

Properties of FB_XmlItemsUtility

Overview

Name	Data type	Access	Description
Result	ET_Result *(see page 29)*	Read	Provides the result of the recently called method or property. A call of this property does not affect the result itself.
IsElementSelected	BOOL	Read	Indicates TRUE if an element is selected. A call of this property returns Ok.
NameOfSelectedElement	STRING[Gc_uiXmlLengthOfString]	Read	Provides the name of the selected element. If the name of the element is a null string, no element is selected. A call of this property returns either Ok or NoElementSelected.
ValueOfSelectedElement	STRING[Gc_uiXmlLengthOfString]	Read/write	Retrieves or sets the value of the selected element. If the returned value is a null string, either no element is selected or the element has no value. Use the property IsElementSelected to validate a null string value. A call of this property returns either Ok or NoElementSelected.
NameOfParentElement	STRING[Gc_uiXmlLengthOfString]	Read	Provides the name of the parent element of the selected element. If the name of the parent element is a null string, either no element is selected or the element is the root element and has no parent. Use the property IsElementSelected to validate a null string value. A call of this property returns either Ok, ElementNotFound, or NoElementSelected.
NumberOfParentElements	INT	Read	Provides the number of parent elements of the selected element. If the returned value is -1, no element is selected. A call of this property returns either Ok or NoElementSelected.
NumberOfChildElements	INT	Read	Provides the number of child elements of the selected element. If the returned value is -1, no element is selected. A call of this property returns either Ok or NoElementSelected.
NumberOfAttributes	INT	Read	Provides the number of the attributes of the selected element. If the returned value is -1, no element is selected. A call of this property returns either Ok or NoElementSelected.

Chapter 12
XML Functions

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
12.1	FC_XmlGetElementValue	102
12.2	FC_XmlSetElementValue	105

Section 12.1
FC_XmlGetElementValue

What Is in This Section?

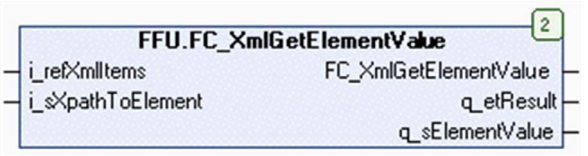
This section contains the following topics:

Topic	Page
FC_XmlGetElementValue Functional Description	103
FC_XmlGetElementValue Considerations	104

FC_XmlGetElementValue Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-



Functional Description

The function FC_XmlGetElementValue is used to read the value of the specified XML element from the buffer of type XmlItems.

The return value is TRUE if the function has been executed successfully. If the return value is FALSE, evaluate the output q_etResult.

Consider the use of the function block FB_XmlItemsUtilities rather than this legacy function. It provides additional features to obtain information from the array of type XmlItems.

Interface

Input	Data type	Description
i_refXmlItems	REFERENCE TO XmlItems	Buffer provided by the application which contains the elements and attributes read from or to be written to an XML file.
i_sXpathToElement	STRING[255]	XPath expression to specify the element that is to be read.

Output	Data type	Description
q_etResult	ET_Result	Provides diagnostic information as a numeric value.
q_sElementValue	STRING[Gc_uiXmlLengthOfString]	Provides the value of the specified element.

XPath Expressions

Use the syntax of the XPath (XML Path) language to specify the element to be selected.

The table lists the supported XPath expressions:

XPath expression	Description
`/…/<elementname>`	Selects the first element matching the specified path.
`/…/<elementname>[<n>]`	Selects the nth element matching the specified path.
`/…/<elementname>[@<attribute>]`	Selects the first element matching the specified path that has the specified attribute.
`/…/<elementname>[@<attribute>=<value>]`	Selects the first element matching the specified path that has the specified attribute and value.

The predicates (the expressions within square brackets `[]`) can be followed by a slash `/` together with an element name to address the next child element.

Example: `/…/<elementname>[<n>]/<elementname>`

FC_XmlGetElementValue Considerations

Considerations for Reading XML Values

Consider the following constraints:

- The execution of the function or the search of the specified element in the buffer of type `XmlItems` is a time-consuming process and can take up to several milliseconds. To help prevent other processes from being blocked by this function, create a separate task with low priority (>24) for it. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- Depending on the size of the buffer `XmlItems` and the position where the element is stored, the execution time of this function can vary from one call to another. Consider this during the commissioning of your application.
- The function is designed to process exactly one element. If your buffer contains more than one element matching the XPath expression, it processes the element that is found first.

Section 12.2
FC_XmlSetElementValue

What Is in This Section?

This section contains the following topics:

Topic	Page
FC_XmlSetElementValue Functional Description	106
FC_XmlSetElementValue Considerations	107

FC_XmlSetElementValue Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-



Functional Description

The function FC_XmlSetElementValue is used to modify the value of the specified XML element from the buffer of type XmlItems.

The return value is TRUE if the function has been executed successfully. If the return value is FALSE, evaluate the output q_etResult.

Consider the use of the function block FB_XmlItemsUtilities rather than this function. It provides additional features to modify data in the array of type XmlItems.

Interface

Input	Data type	Description
i_refXmlItems	REFERENCE TO XmlItems	Buffer provided by the application which contains the elements and attributes read from or to be written to an XML file.
i_sXpathToElement	STRING[255]	XPath expression to specify the element that is to be read.
i_sElementValue	STRING[Gc_uiXmlLengthOfString]	The value to be set for the specified element.

Output	Data type	Description
q_etResult	ET_Result	Provides diagnostic information as a numeric value.

XPath Expressions

Use the syntax of the XPath (XML Path) language to specify the element to be selected.

The table lists the supported XPath expressions:

XPath expression	Description
`/…/<elementname>`	Selects the first element matching the specified path.
`/…/<elementname>[<n>]`	Selects the nth element matching the specified path.
`/…/<elementname>[@<attribute>]`	Selects the first element matching the specified path that has the specified attribute.
`/…/<elementname>[@<attribute>=<value>]`	Selects the first element matching the specified path that has the specified attribute and value.

The predicates (the expressions within square brackets `[]`) can be followed by a slash `/` together with an element name to address the next child element.

Example: `/…/<elementname>[<n>]/<elementname>`

FC_XmlSetElementValue Considerations

Considerations for Setting XML Values

Consider the following constraints:

- The execution of the function or the search of the specified element in the buffer of type `XmlItems` is a time-consuming process and can take up to several milliseconds. To help prevent other processes from being blocked by this function, create a separate task with low priority (>24) for it. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- Depending on the size of the buffer `XmlItems` and the position where the element is stored, the execution time of this function can vary from one call to another. Consider this during the commissioning of your application.
- The function is designed to process exactly one element. If your buffer contains more than one element matching the XPath expression, it processes the element that is found first.

Part VII
CSV Program Organization Units (POU)

Chapter 13
CSV Function Blocks

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
13.1	FB_CsvRead	112
13.2	FB_CsvWrite	120

Section 13.1
FB_CsvRead

What Is in This Section?

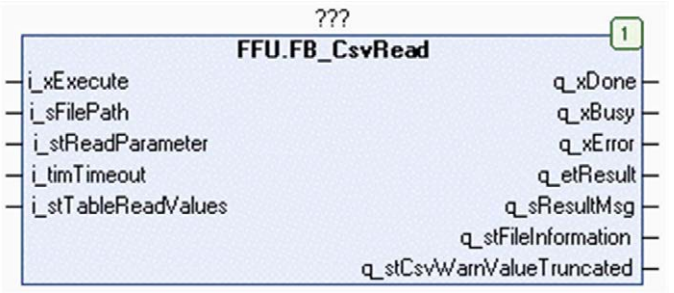
This section contains the following topics:

Topic	Page
FB_CsvRead Functional Description	113
FB_CsvRead Considerations	116
FB_CsvRead Troubleshooting	117
FB_CsvRead Example	118

FB_CsvRead Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-



Functional Description

The function block FB_CsvRead is used to read a CSV file located on the file system of the controller or on the extended memory (for example, an SD memory card). For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

The CSV file to read contains a number of values (columns) which are arranged in single records (rows). The values are separated by a specific delimiter. The records are separated by a line break.

Based on the specified character code for the delimiter, the function block identifies the individual values while reading the content of the file. The character code for the line break depends on the operating system where the file has been created. The function block supports the three most commonly used line break character codes (ASCII). It detects the line break character used while reading the content of the file.

The following line break characters (ASCII) are supported:

- CRLF (0D0A hex): Used on operating systems such as Windows and MS-DOS.
- LF (0A hex): Used on operating systems such as Unix, Linux, Mac OS X, and Android.
- CR (0D hex): Used on operating systems such as Mac OS X version 9 or earlier.

The values read from the specified file are stored in the read buffer provided by the application in variables of type STRING. Declare the read buffer in the application as a two-dimensional ARRAY of type STRING. Use the input `i_stTableReadValues` to provide the dimensions of the array and the pointer to that array to the function block. For further information, refer to the structure `ST_CsvTable` *(see page 37)*.

When executing the function block, the input `i_stTableWriteValues.pbyTable` is stored internally for further use. In case an online change event is detected while the function block is executed (`q_xBusy` = TRUE), the internally used variables are updated with the present value of the input.

NOTE: Do not reassign the `i_stTableWriteValues.pbyTable` to a different memory area while the function block is executed.

The file to be read must contain only ASCII characters to help ensure the correct rendition of the content of the file in the application. Files can contain a byte order mark (BOM) at the beginning which indicates the encoding of the processed file. ASCII-encoded files do not contain a BOM. The function block verifies the specified file for specific BOMs.

In case the file contains one of the following BOMs, the execution of the function block is canceled and an error is indicated:

`FE FF hex`, `FF EF hex`, or `EF BB BF hex`

Use the input `i_stReadParameter` to determine the amount of data to be read. You can read the entire content of the file. You can also select a single record (row), a single column, or a single value to be read. Further, you can select to read only the file information which is provided by the output `q_stFileInformation`.

Interface

Input	Data type	Description
`i_xExecute`	BOOL	The function block executes the read operation with the specified CSV file upon a rising edge of this input. Also refer to the chapter *Behavior of Function Blocks with the Input i_xExecute* *(see page 21)*.
`i_sFilePath`	STRING[255]	File path to the CSV file. If a file name is specified without file extension, the function block adds the extension .csv.
`i_stReadParameter`	`ST_CsvReadParameter`	Specifies the content to be read from the file. Refer to `ST_CsvReadParameter` *(see page 41)*.
`i_timTimeout`	TIME	After this time has elapsed, the execution is canceled. If the value is T#0s, the default value T#2s is applied.
`i_stTableReadValues`	`ST_CsvTable`	Structure to pass the buffer provided by the application to the function block (refer to the `ST_CsvTable` structure *(see page 37)*).

Output	Data type	Description
q_xDone	BOOL	If this output is set to TRUE, the execution has been completed successfully.
q_xBusy	BOOL	If this output is set to TRUE, the function block execution is in progress.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to q_etResult and q_etResultMsg.
q_etResult	ET_Result	Provides diagnostic and status information as a numeric value. If q_xBusy = TRUE, the value indicates the status. If q_xDone or q_xError = TRUE, the value indicates the result.
q_sResultMsg	STRING[80]	Provides additional diagnostic and status information as a text message.
q_stFileInformation	ST_CsvFileInfo	The structure contains information about the last file that has been processed
q_stWarnValueTruncated	ST_CsvWarnValueTruncated	Provides information about the first value that has been truncated, if available. **NOTE:** The output is updated along with q_xDone.

For more information about the signal behavior of the basic inputs and outputs, refer to the chapter *Behavior of Function Blocks with the Input i_xExecute (see page 21)*.

Usage of Variables of Type POINTER TO … or REFERENCE TO …

The function block provides inputs and/or in/outputs of type POINTER TO… or REFENCE TO…. With the use of such a pointer or reference, the function block accesses the addressed memory area. In case of an online change event, it may happen that memory areas are moved to new addresses and in consequence a pointer or reference becomes invalid. To prevent errors associated with invalid pointers, variables of type POINTER TO… or REFERENCE TO… must be updated cyclically or at least at the beginning of the cycle in which they are used.

FB_CsvRead Considerations

Considerations

Consider the following constraints for reading a CSV file:

- Only CSV files with ASCII encoding are supported.
  It is not verified whether the file encoding is actually ASCII. If the file contains characters outside the range of the ASCII character code set, the values rendered by the application are not valid.
- The values read from the file are stored as STRING values in the application. This applies even to numeric values. Before they can be processed, the values need to be converted to the appropriate datatype. To achieve this, it is a good practice to use the `STRING_TO_` conversion functions *(see EcoStruxure Machine Expert, Programming Guide)*. The conversion functions require a specific syntax of the STRING values depending on the target datatype. Consider these requirements when creating the files to simplify the processing of read values.
- File operations are time-consuming processes. To help prevent influencing time-critical control functions in your application, create a separate task with lower priority for these processes. For more information about task management, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- The timeout parameter `i_timTimeout` is used to monitor the read operation. If the value specified for the timeout is exceeded during execution of the `FB_CsvRead` function block, the read operation is canceled and the function block generates an error. To select an appropriate value for the timeout parameter, consider that the read operation takes several task cycles. Thus, calculate the product of the number of required task cycles and the task interval to specify the minimum value for the timeout parameter.
- The number of cycles depends, besides the size of the file to be read, on the size of the processing block which can be specified by the parameter `Gc_uiCsvReadProcessingBlockSize` in the GPL *(see page 49)*. Analyzing the content of the file is split into several read operations to reduce the load for the single task cycle. During each read operation, one data block is processed and, if requested, stored to the buffer. The larger the processing block size, the fewer cycles are required for reading the file. However, a larger block size increases the execution time of this single read operation.
- The character code specified as delimiter is not supported as part of a value.
- Do not process the data from the buffer provided via `i_stTableReadValues.pbyTable` as long as the output `q_xBusy` of function block indicates TRUE.
- Do not access the same file with different function blocks at the same time. Use the output `q_xBusy` to lock the execution of different function blocks.

FB_CsvRead Troubleshooting

Troubleshooting

The table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified directory is not available. • The given file path has invalid syntax.	• Verify that the directory exists. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead.
Execution ends with a detected error and result indicates `TableInvalid`.	• The pointer to the read buffer is not assigned.	• Verify the correct assignment of the parameter `pbyTable` in the structure `i_stTableReadValues` *(see page 37)*.
Execution ends with a detected error and result indicates `TableReadValuesInvalid`.	• The dimensions of the specified buffer (table) are not consistent.	• Verify the correct assignment of the parameter in the structure `i_stTableReadValues` *(see page 37)*.
Execution ends with a detected error and result indicates `ReadParameterInvalid`.	• The parameters specified for the read mode are inconsistent. • No character has been specified for the delimiter.	• Verify that the value selected for `etReadMode` is supported by `ET_CsvReadMode` *(see page 27)*. • If `etReadMode` = `OneRow`, the value for `uiNumOfRow` must not be 0. • If `etReadMode` = `OneColumn`, the value for `uiNumOfColumn` must not be 0. • If `etReadMode` = `OneValue`, the values for `uiNumOfRow` and `uiNumOfColumn` must not be 0. • The value for `sDelimiter` must not be empty.
Execution ends with a detected error and result indicates `Timeout`.	• The specified value for the timeout is too low.	• Increase the timeout parameter according to the number of cycles needed to read the file. Also consider the task interval. • Increase the processing block size per read operation to reduce the number of function block calls which are needed to read the file.

FB_CsvRead Example

Overview

The following example shows how the values read from a CSV file are stored in the buffer provided by the application for further processing.

Example CSV File

Content of the file example.csv with delimiter specified as semicolon(;):

```
PNo;PName;PValue;PUnit;Description
1;Velo_Max;1500;rpm;maximum velocity of the motor
2;Velo_Min;100;rpm;minimum velocity of the motor
3;Velo_ManMode_Slow;150;rpm;velocity manual mode slow move
4;Velo_ManMode_Fast;600;rpm;velocity manual mode fast move
```

Example Program

```
PROGRAM SR_Example
VAR
  fbRead :FFU.FB_CsvRead;
  xCmdRead :BOOL;
  asCsvTable:ARRAY[0..c_uiNumOfRows-1,0..c_uiNumOfColumns-
1] OF STRING(c_uiLengthOfValue);
  stCsvTable: FFU.ST_CsvTable;
END_VAR
VAR CONSTANT
  c_uiNumOfRows :UINT:= 8;
  c_uiNumOfColumns :UINT:= 7;
  c_uiLengthOfValue :UINT:= 20;
END_VAR

fbRead.i_stTableReadValues.pbyTable := ADR(asCsvTable);
fbRead.i_stTableReadValues.uiNumOfRows := c_uiNumOfRows;
fbRead.i_stTableReadValues.uiNumOfColumns := c_uiNumOfColumns;
fbRead.i_stTableReadValues.udiSizeOfTable := SIZEOF(asCsvTable);

fbRead.i_stReadParameter.sDelimiter := ';';
fbRead.i_stReadParameter.etReadMode := FFU.ET_CsvReadMode.AllValues;
fbRead.i_stReadParameter.udiNumOfRow := 0;
fbRead.i_stReadParameter.udiNumOfColumn := 0;

fbRead(
       i_xExecute:= xCmdRead,
       i_sFilePath:= '/sd0/Example.csv',
       i_stTableReadValues:= ,
```

```
        i_stReadParameter:= ,
        i_timTimeout:= ,
        q_xDone=> ,
        q_xBusy=> ,
        q_xError=> ,
        q_etResult=> ,
        q_sResultMsg=> ,
        q_stFileInformation=> ,
        q_stCsvWarnValueTruncated=> ,
        xValueTruncated=> );
```

Buffer

The buffer provided by the application contains the elements read from the CSV file:

Array Index	0	1	2	3	4	5	6
0	PNo	PName	PValue	PUnit	Description	–	–
1	1	Velo_Max	1500	rpm	maximum velocity of	–	–
2	2	Velo_Min	100	rpm	minimum velocity of	–	–
3	3	Velo_ManMode_Slow	150	rpm	velocity manual mode	–	–
4	4	Velo_ManMode_Fast	600	rpm	velocity manual mode	–	–
5	–	–	–	–	–	–	–
6	–	–	–	–	–	–	–
7	–	–	–	–	–	–	–

NOTE: The values in the column Description have been truncated while reading the file because the length of the STRINGs in the read buffer is limited to 20 characters. The output `q_stWarn-ValueTruncated` of the function block indicates that the value in row 2 and column 5 has been truncated.

Section 13.2
FB_CsvWrite

What Is in This Section?

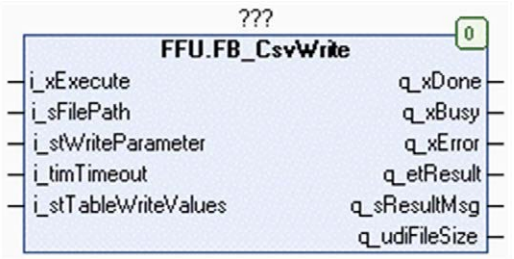
This section contains the following topics:

Topic	Page
FB_CsvWrite Functional Description	121
FB_CsvWrite Considerations	123
FB_CsvWrite Troubleshooting	124
FB_CsvWrite Example	125

FB_CsvWrite Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-



Functional Description

The function block FB_CsvWrite is used to write values to a CSV file located on the file system of the controller or on the extended memory (for example, an SD memory card). It can also create a new file. For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

The data to be written to the file is stored in the buffer provided by the application as variables of type STRING. Declare the buffer in the application as a two-dimensional ARRAY of type STRING. Use the input i_stTableWriteValues to provide the dimensions of the array and the pointer to that array to the function block. For further information, refer to the structure ST_CsvTable *(see page 37)*.

When executing the function block, the input i_stTableWriteValues.pbyTable is stored internally for further use. In case an online change event is detected while the function block is executed (q_xBusy = TRUE), the internally used variables are updated with the present value of the input.

NOTE: Do not reassign the i_stTableWriteValues.pbyTable to a different memory area while the function block is executed.

The two-dimensional array represents the table structure consisting of rows and columns. Each row represents a record. The number of columns represents the maximum number of values one record can have.

The input `i_stWriteParameter` provides the parameter to control the write operation. Use the parameter `sDelimiter` to specify the character code for the delimiter that is inserted to separate the individual values of the file. The value of the parameter `etModeFileOpen` allows you to specify whether the data is to be appended to an existing file or whether a new file is to be created. Use the parameters `uiNumOfRows` and `uiNumOfColumns` to specify the amount of data to be written.

The character code `LF (0A hex)` is inserted to provoke a line break between two records.

Interface

Input	Data type	Description
`i_xExecute`	BOOL	The function block opens or creates the specified CSV file and writes the specified content into it upon a rising edge of this input.
`i_sFilePath`	STRING[255]	File path to the CSV file. If a file name is specified without file extension, the function block adds the extension .csv.
`i_stWriteParameter`	`ST_WriteParameter`	Specifies the mode for opening the CSV file and the content to be written into the file.
`i_timTimeout`	TIME	After this time has elapsed, the execution is canceled. If the value is T#0s, the default value T#2s is applied.
`i_stTableWriteValues`	`ST_CsvTable`	Structure to pass the buffer provided by the application to the function block (refer to the `ST_CsvTable` structure *(see page 37)*).

Output	Data type	Description
`q_xDone`	BOOL	If this output is set to TRUE, the execution has been completed successfully.
`q_xBusy`	BOOL	If this output is set to TRUE, the function block execution is in progress.
`q_xError`	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to `q_etResult` and `q_etResultMsg`.
`q_etResult`	`ET_Result`	Provides diagnostic and status information as a numeric value. If `q_xBusy` = TRUE, the value indicates the status. If `q_xDone` or `q_xError` = TRUE, the value indicates the result.
`q_sResultMsg`	STRING[80]	Provides additional diagnostic and status information as a text message.
`q_udiFileSize`	UDINT	Provides the file size in bytes of the file recently processed.

Usage of Variables of Type POINTER TO … or REFERENCE TO …

The function block provides inputs and/or in/outputs of type POINTER TO… or REFENCE TO…. With the use of such a pointer or reference, the function block accesses the addressed memory area. In case of an online change event, it may happen that memory areas are moved to new addresses and in consequence a pointer or reference becomes invalid. To prevent errors associated with invalid pointers, variables of type POINTER TO… or REFERENCE TO… must be updated cyclically or at least at the beginning of the cycle in which they are used.

FB_CsvWrite Considerations

Considerations

Consider the following constraints for writing a CSV file:

- File operations are time-consuming processes. To help prevent influencing time-critical control functions in your application, create a separate task with lower priority for these processes. For more information about task management, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- The timeout parameter i_timTimeout is used to monitor the file operation. If the value specified for the timeout is exceeded during execution of the function block, the write operation is canceled and the function block indicates an error. To select an appropriate value for the timeout parameter, consider that the file operation takes several task cycles. Thus, calculate the product of the number of required task cycles and the task interval to specify the minimum value for the timeout parameter.
- The number of cycles depends, besides the amount of data to be written, on the size of the processing block which can be specified by the parameter Gc_uiCsvWriteProcessing-BlockSize in the GPL *(see page 49)*. The writing data to the file is split into several write operations to reduce the load for the single task cycle. During each write operation, one data block is processed and written to the file. The larger the processing block size the fewer cycles are required for creating and writing the file. However, a larger block size increases the execution time of each write operation.
- Do not process the data from the buffer provided via i_stTableReadValues.pbyTable as long as the output q_xBusy of function block indicates TRUE.
- Do not access the same file with different function blocks at the same time. Use the output q_xBusy to lock the execution of different function blocks.

FB_CsvWrite Troubleshooting

Troubleshooting

This table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified directory is not available. • The given file path has invalid syntax. • The specified file does not exist on the specified file path.	• Verify that the directory exists. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead. • For the parameter `etModeFileOpen` *(see page 28)*, select the value `AppendPlus`.
Execution ends with a detected error and result indicates `FileAlreadyExists`.	• The specified file already exists but the `etModeFileOpen` *(see page 28)* does not allow overwriting the file.	• Specify another file name. • If the existing file can be overwritten, select for the parameter `etModeFileOpen` *(see page 28)* the value `CreatePlus`.
Execution ends with a detected error and result indicates `TableInvalid`.	• The pointer to the read buffer is not assigned.	• Verify the correct assignment of the parameter `pbyTable` in the structure `i_stTableWriteValues` *(see page 37)*.
Execution ends with a detected error and result indicates `TableWriteValuesInvalid`.	• The dimensions of the specified CSV table are not consistent.	• Verify the correct assignment of the parameter in the structure `i_stTableWriteValues` *(see page 37)*.
Execution ends with a detected error and result indicates `WriteParameterInvalid`.	• The parameter specified for the write mode are inconsistent. • No character has been specified for the delimiter.	• Verify that the selected value for `etModeFileOpen` is supported by `etModeFileOpen` *(see page 28)* • The value for `sDelimiter` must not be empty.
Execution ends with a detected error and result indicates `Timeout`.	• The specified value for the timeout is too low.	• Increase the timeout parameter according to the number of cycles needed to create the file. Consider also the task interval. • Increase the processing block size per write operation to reduce the number of function block calls which are needed to create the file.

FB_CsvWrite Example

Overview

The following example shows how to implement the FB_CsvWrite function block to write data to an existing CSV file.

Buffer

The buffer provided by the application contains the elements to be written to the CSV file:

Array Index	0	1	2	3	4	5	6
0	2017-02-24	09:30:10	OpMode	Machine - AutoMode	–	–	–
1	2017-02-24	09:35:27	Error	Module1 - ErrorID 16#A123	–	–	–
2	2017-02-24	09:35:27	State	Machine - ErrorStop	–	–	–
3	–	–	–	–	–	–	–
4	–	–	–	–	–	–	–
5	–	–	–	–	–	–	–
6	–	–	–	–	–	–	–
7	–	–	–	—	–	–	–

Example Program

```
PROGRAM SR_Example
VAR
  fbWrite :FFU.FB_CsvWrite;
  xCmdWrite :BOOL;
  asCsvTable:ARRAY[0..c_uiNumOfRows-1,0..c_uiNumOfColumns-
1] OF STRING(c_uiLengthOfValue);
  stCsvTable: FFU.ST_CsvTable;
END_VAR
VAR CONSTANT
  c_uiNumOfRows :UINT:= 8;
  c_uiNumOfColumns :UINT:= 7;
  c_uiLengthOfValue :UINT:= 80;
END_VAR

fbWrite.i_stTableWriteValues.pbyTable := ADR(asCsvTable);
fbWrite.i_stTableWriteValues.uiNumOfRows := c_uiNumOfRows;
fbWrite.i_stTableWriteValues.uiNumOfColumns := c_uiNumOfColumns;
```

```
fbWrite.i_stTableWriteValues.udiSizeOfTable := SIZEOF(asCsvTable);

fbWrite.i_stWriteParameter.sDelimiter := ';';
fbWrite.i_stWriteParameter.etModeFileOpen := FFU.ET_ModeFileOpen.Append
;
fbWrite.i_stWriteParameter.udiNumOfRow := 3;
fbWrite.i_stWriteParameter.udiNumOfColumn := c_uiNumOfColumns;

fbWrite(
       i_xExecute:= xCmdWrite,
       i_sFilePath:= '/sd0/Example.csv',
       i_stWriteParameter:= ,
       i_timTimeout:= ,
       i_stTableWriteValues:= ,
       q_xDone=> ,
       q_xBusy=> ,
       q_xError=> ,
       q_etResult=> ,
       q_sResultMsg=> ,
       q_udiFileSize=> );
```

Example CSV File

Content of example.csv after appending new data:

```
Date;Time;Eventype;Event;;;;
2017-02-24;09:27:36;OpMode;Machine - Power on;;;;
2017-02-24;09:27:54;OpMode;Machine - Manual Mode;;;;
2017-02-24;09:28:32;OpMode;Machine - Homing;;;;
2017-02-24;09:29:44;State;Machine - All Modules Homed;;;;
2017-02-24;09:30:10;OpMode;Machine - AutoMode;;;;
2017-02-24;09:35:27;Error;Module1 - ErrorID 16#A123;;;;
2017-02-24;09:35:27;State;Machine - ErrorStop;;;;
```

Part VIII
JSON Program Organization Units (POU)

Chapter 14
JSON Function Blocks

Section 14.1
FB_CreateJsonFormattedString

What Is in This Section?

This section contains the following topics:

Topic	Page
FB_CreateJsonFormattedString Functional Description	130
New (Method)	131
Copy (Method)	131
ObjectOpen (Method)	132
ObjectClose (Method)	133
AddNameValuePairObject (Method)	134
AddNameValuePairArray (Method)	135
AddNameValuePair (Method)	136
AddNameValuePair<data type> (Method)	137
AddNameValuePairNull (Method)	139
ArrayAddArray (Method)	140
ArrayAddValue (Method)	140
ArrayAdd<data type> (Method)	142
ArrayAddNull (Method)	143
ArrayClose (Method)	144
Properties of FB_CreateJsonFormattedString	145
FB_CreateJsonFormattedString Example	146

FB_CreateJsonFormattedString Functional Description

Overview

Type:	Function block
Available as of:	V1.2.0.3
Inherits from:	-
Implements:	-

Functional Description

The function block `FB_CreateJsonFormattedString` is used to create a JSON-formatted STRING. The created STRING may be processed only as ASCII coded text string.

The function block does not provide any parameters, but methods and properties to control the creation process and to monitor the status. The instance of the function block is used exclusively for holding the local variables which are accessed during the sequential processing of the associated methods or properties. Therefore, it is not required to call the function block in the application code.

New (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Creates a new STRING that contains a single pair of curly brackets `{}`.

When you call the method `New`, an existing STRING is overwritten.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Example

Calling the method `New` creates the following STRING:

```
{}
```

Copy (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Copies the present JSON-formatted STRING to the specified buffer in the application.

The return value is of data type UDINT and indicates the number of bytes that have been copied to the specified buffer.

Evaluate the property `Result`, in case the return value is `0`. If the size of the specified buffer is less than the length of the present JSON-formatted STRING, no data is copied.

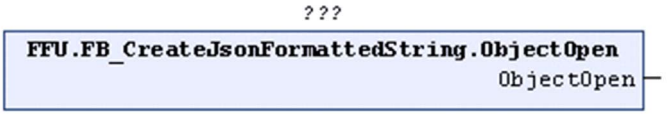
Interface

Input	Data type	Description
`i_anyBuffer`	ANY	Buffer provided by the application in which the data shall be copied. Data types ARRAY and STRING of appropriate size are supported.

`ObjectOpen` (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Inserts a left curly bracket to an open ARRAY before the last right curly bracket of the STRING that is being processed. According to the definition of the JSON syntax, the left curly bracket indicates the start of an object.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The maximum length of the present STRING is reached.	The STRING remains unchanged.
The maximum number of levels is reached for the present STRING.	The STRING remains unchanged.

Example

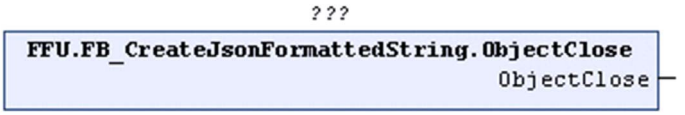
Calling the method `ObjectOpen` adds the left curly bracket, marked in bold magenta in the example, to the STRING:

```
{"Array":[{}
```

`ObjectClose` (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Inserts a right curly bracket before the last right curly bracket of the STRING that is being processed. According to the definition of the JSON syntax, the right curly bracket indicates the end of an object. After the method `ObjectClose` has been called, the corresponding object is closed and no more name/value pairs can be added.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The present STRING does not contain an open object.	The STRING remains unchanged.
The maximum length of the present STRING is reached.	The STRING remains unchanged.

Example

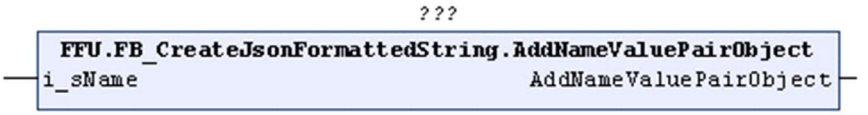
Calling the method `ObjectClose` adds the right curly bracket, marked in bold magenta in the example, to the STRING:

```
{"Key":1,"Object":{"ID":1, "Version":"5.0"}}
```

`AddNameValuePairObject` (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Adds a name/value pair to the STRING that is being processed with the value being of type object.

After the method `AddNameValuePairObject` has been called, the object which represents the value that corresponds to the specified name is open.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The maximum length of the present STRING is reached.	The STRING remains unchanged.
The maximum number of levels is reached for the present STRING.	The STRING remains unchanged.

Interface

Input	Data type	Description
`i_sName`	STRING(`GPL.Gc_uiJsonMaxLengthOfName`)	Specifies the name of the name/value pair to be added. The quotation marks surrounding the <name> must not be specified explicitly, they are implicitly added by the method.

Example

Calling the method `AddNameValuePairObject` adds the text marked in bold magenta in the example to the STRING:

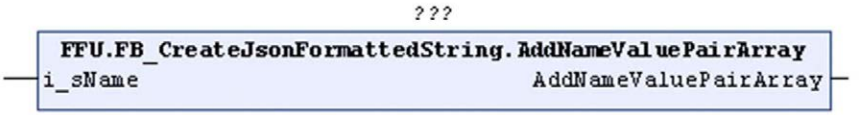
`{"Key":1,"<name>":{}`

<name> corresponds to the value specified with the input `i_sName` of the method.

`AddNameValuePairArray` (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Adds a name/value pair to the STRING that is being processed with the value being of type ARRAY.

After the method `AddNameValuePairArray` has been called, the ARRAY which represents the value that corresponds to the specified name is open.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The maximum length of the present STRING is reached.	The STRING remains unchanged.
The maximum number of levels is reached for the present STRING.	The STRING remains unchanged.

Interface

Input	Data type	Description
`i_sName`	STRING(`GPL.Gc_uiJsonMaxLengthOfName`)	Specifies the name of the name/value pair to be added. The quotation marks surrounding the <name> must not be specified explicitly, they are implicitly added by the method.

Example

Calling the method `AddNameValuePairArray` adds the text marked in bold magenta in the example to the STRING:

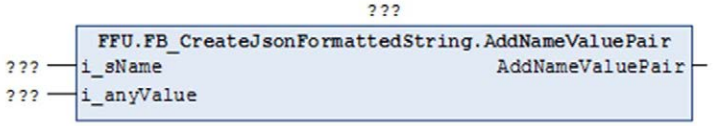
`{"Key":1,"<name>":[}`

<name> corresponds to the value specified with the input `i_sName` of the method.

`AddNameValuePair` (Method)

Overview

Type:	Method
Available as of:	1.2.3.0



Functional Description

Adds a name/value pair to the STRING that is being processed. The format of the value added to the STRING depends on the data type of the variable assigned to the input `i_anyValue`. The assigned value is converted, if required, to an ASCII STRING and added in the suitable format to the STRING that is being processed.

In contrast to the Methods `AddNameValuePair<data type>` which support only one specific data type for the value, this single method supports several data types for the input `i_anyValue`.

Variables of data type BOOL, STRING, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, LWORD, REAL, LREAL, SINT, USINT, LINT, ULINT, TIME, LTIME, DATE_AND_TIME, DATE, and TOD are supported for the input `i_anyValue`.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The maximum length of the present STRING is reached.	The STRING remains unchanged.
The data type (for example ARRAY or WSTRING) of the variable assigned to `i_anyValue` is not supported.	The STRING remains unchanged.

Interface

Input	Data type	Description
i_sName	STRING (GPL.Gc_uiJsonMaxLengthOfName)	Specifies the name of the name/value pair to be added. The quotation marks surrounding the <name> must not be specified explicitly, they are implicitly added by the method.
i_anyValue	ANY*	Specifies the value to be added.
* Supported data types are: BOOL, STRING, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, LWORD, REAL, LREAL, SINT, USINT, LINT, ULINT, TIME, LTIME, DATE_AND_TIME, DATE, and TOD.		

Example

Calling the method AddNameValuePair adds the text marked in bold magenta in the example to the STRING:

```
{"Key":1,"<name>":<value>}
```

<name> corresponds to the value specified with the input i_sName of the method.

<value> corresponds to the value specified with the input i_anyValue of the method.

AddNameValuePair<data type> (Method)

Overview

This section provides a generic description for the following methods:

- AddNameValuePairINT
- AddNameValuePairDINT
- AddNameValuePairUDINT
- AddNameValuePairREAL
- AddNameValuePairSTRING
- AddNameValuePairBOOL

Type:	Method
Available as of:	V1.2.0.3

As an example, the method AddNameValuePairINT is illustrated in the following figure:



Functional Description

Adds a name/value pair to the STRING that is being processed with the value being of the data type that is indicated in the name of the method:

- INT = INTEGER
- DINT = DOUBLE INTEGER
- UDINT = UNSIGNED DOUBLE INTEGER
- REAL = floating point value as per IEC data type REAL
- STRING = STRING
- BOOL = BOOL

The assigned value is converted, if required, to an ASCII STRING and added in the suitable format to the STRING that is being processed.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The maximum length of the present STRING is reached.	The STRING remains unchanged.

Interface

Input	Data type	Description
`i_sName`	STRING(`GPL.Gc_uiJsonMaxLengthOfName`)	Specifies the name of the name/value pair to be added. The quotation marks surrounding the `<name>` must not be specified explicitly, they are implicitly added by the method.
`i_*Value`	*	Specifies the value to be added.
* Data type that corresponds to the method used.		

Example

Calling the method `AddNameValuePair<data type>` adds the text marked in bold magenta in the example to the STRING:

`{"Key":1,"<name>":<value>}`

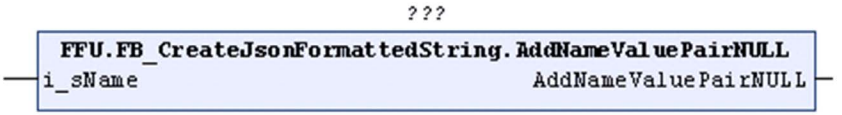
`<name>` corresponds to the value specified with the input `i_sName` of the method.

`<value>` corresponds to the value specified with the input `i_*Value` of the method.

AddNameValuePairNull (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Adds a name/value pair to the STRING that is being processed with the value being the literal name token `null`.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The maximum length of the present STRING is reached.	The STRING remains unchanged.

Interface

Input	Data type	Description
`i_sName`	STRING(`GPL.Gc_uiJsonMaxLengthOfName`)	Specifies the name of the name/value pair to be added. The quotation marks surrounding the <name> must not be specified explicitly, they are implicitly added by the method.

Example

Calling the method `AddNameValuePairNull` adds the text marked in bold magenta in the example to the STRING:

```
{"Key":1,"<name>":null}
```

<name> corresponds to the value specified with the input `i_sName` of the method.

ArrayAddArray (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Opens an ARRAY within an open ARRAY in the STRING that is being processed.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The present STRING does not contain an open ARRAY.	The STRING remains unchanged.
The maximum length of the present STRING is reached.	The STRING remains unchanged.
The maximum number of levels is reached for the present STRING.	The STRING remains unchanged.

Example

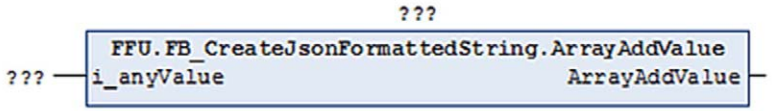
Calling the method `ArrayAddArray` adds the left square bracket, marked in bold magenta in the example, to the STRING:

```
{"Array":[[}
```

ArrayAddValue (Method)

Overview

Type:	Method
Available as of:	1.2.3.0



Functional Description

Adds a value to an open ARRAY in the STRING that is being processed.

The format of the value added to the STRING depends on the data type of the variable assigned to the input `i_anyValue`. The assigned value is converted, if required, to an ASCII STRING and added in the suitable format to the STRING that is being processed.

In contrast to the Methods `ArrayAdd<data type>` which support only one specific data type for the value, this single method supports several data types for the input `i_anyValue`.

Variables of data type BOOL, STRING, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, LWORD, REAL, LREAL, SINT, USINT, LINT, ULINT, TIME, LTIME, DATE_AND_TIME, DATE, and TOD are supported for the input `i_anyValue`.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The present STRING does not contain an open ARRAY.	The STRING remains unchanged.
The maximum length of the present STRING is reached.	The STRING remains unchanged.
The data type (for example ARRAY or WSTRING) of the variable assigned to `i_anyValue` is not supported.	The STRING remains unchanged.

Interface

Input	Data type	Description
`i_anyValue`	ANY*	Specifies the value to be added.
* Supported data types are: BOOL, STRING, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, LWORD, REAL, LREAL, SINT, USINT, LINT, ULINT, TIME, LTIME, DATE_AND_TIME, DATE, and TOD.		

Example

Calling the method `ArrayAddValue` adds the text marked in bold magenta in the example to the STRING:

```
{"Array":[1,2,<value>}
```

<value> corresponds to the value specified with the input `i_anyValue` of the method.

ArrayAdd<data type> (Method)

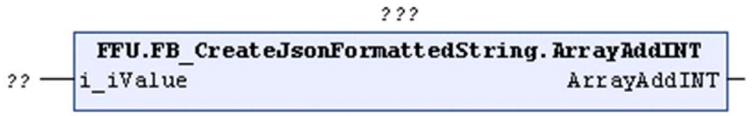
Overview

This section provides a generic description for the following methods:

- ArrayAddINT
- ArrayAddDINT
- ArrayAddUDINT
- ArrayAddREAL
- ArrayAddSTRING
- ArrayAddBOOL

Type:	Method
Available as of:	V1.2.0.3

As an example, the method ArrayAddINT is illustrated in the following figure:



Functional Description

Adds a value to an open ARRAY in the STRING that is being processed with the value being of the data type that is indicated in the name of the method:

Datatype	Prefix	Description
INT	i	INTEGER
DINT	di	DOUBLE INTEGER
UDINT	udi	UNSIGNED DOUBLE INTEGER
REAL	r	Floating point value as per IEC data type REAL
STRING	s	STRING
BOOL	x	BOOL

The assigned value is converted, if required, to an ASCII STRING and added in the suitable format to the STRING that is being processed.

The return value is TRUE if the function was executed successfully. Evaluate the property Result, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The present STRING does not contain an open ARRAY.	The STRING remains unchanged.
The maximum length of the present STRING is reached.	The STRING remains unchanged.

Interface

Input	Data type	Description
`i_*Value`	*	Specifies the value to be added.
* Data type and prefix that corresponds to the method used.		

Example

Calling the method `ArrayAdd<data type>` adds the text marked in bold magenta in the example to the STRING:

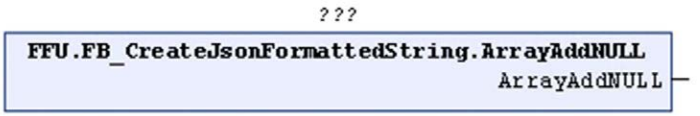
`{"Array":[1,2,<value>}`

`<value>` corresponds to the value specified with the input `i_*Value` of the method.

`ArrayAddNull` (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Adds the literal name token `null` to an open ARRAY in the STRING that is being processed.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The present STRING does not contain an open ARRAY.	The STRING remains unchanged.
The maximum length of the present STRING is reached.	The STRING remains unchanged.

Example

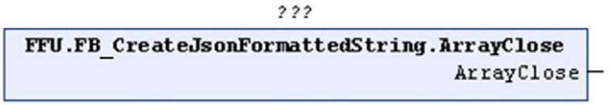
Calling the method `ArrayAddNull` adds the text marked in bold magenta in the example to the STRING:

```
{"Array":[1,2,null}
```

`ArrayClose` (Method)

Overview

Type:	Method
Available as of:	V1.2.0.3



Functional Description

Inserts a right square bracket before the last right curly bracket of the STRING that is being processed. According to the definition of the JSON syntax, the right square bracket indicates the end of an ARRAY. After the method `ArrayClose` has been called, the ARRAY is closed and no more values can be added to the corresponding ARRAY.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Unsuccessful execution of the method can have the following causes:

Possible Cause	Effect
The present STRING does not contain an open ARRAY.	The STRING remains unchanged.
The maximum length of the present STRING is reached.	The STRING remains unchanged.

Example

Calling the method `ArrayClose` adds the right square bracket, marked in bold magenta in the example, to the STRING:

```
{"Array":[1,2,3]}
```

Properties of `FB_CreateJsonFormattedString`

Overview

Name	Data type	Access	Description
`PointerToJsonString`	POINTER TO BYTE	Read	Provides a pointer to the JSON-formatted STRING that is being processed.
`LengthOfJsonString`	UDINT	Read	Provides the present length (number of characters) of the JSON-formatted STRING that is being processed.
`Result`	`ET_Result`	Read	Indicates the result of the recently called method.
`IsJsonStringValid`	BOOLEAN	Read	Indicates if the JSON-formatted STRING that is being processed is valid. Indicates TRUE if the STRING meets the following conditions: ● The STRING is not empty. ● The STRING does not contain an open object. ● The STRING does not contain an open ARRAY.
`IsArrayOpen`	BOOLEAN	Read	If TRUE, the JSON-formatted STRING that is being processed contains an open ARRAY.
`IsObjectOpen`	BOOLEAN	Read	If TRUE, the JSON-formatted STRING that is being processed contains an open object.

FB_CreateJsonFormattedString Example

Overview

The following example shows how to build a text STRING in JSON format by using the function block FB_CreateJsonFormattedString. After the text STRING has been created, it is stored in a new file on the file system of the controller by using the function block FB_WriteFile.

Example Program

```
//JSON Example - Declaration Part
PROGRAM SR_Main
VAR
    xStart : BOOL;
    xReset : BOOL;
    etResult : FFU.ET_Result;
    xJsonStringInvalid : BOOL;
    xProcessCompleted : BOOL;
    sResultMsg : STRING(80):= 'Ready for start';
    xErrorWriteFile : BOOL;
    dtTimestamp : DATE_AND_TIME;
    udiMessageId : UDINT := 12587;
    rTotalEnergy : REAL    := 35.6;
    rMaxVoltage : REAL    := 402.56;
    rLowVoltage : REAL    := 399.03;
    rMaxCurrent : REAL    := 18.459;
    rTemperatureSensor1 : REAL    := 21.65;
    rTemperatureSensor2 : REAL    := 32.47;
    rTemperatureSensor3 : REAL    := 36.52;
    rTemperatureSensor4 : REAL    := 47.35;
    iState : INT;
    xErr : BOOL;
    xStartOld : BOOL;
    udiResult : UDINT;
    sJsonString : STRING(FFU.GPL.Gc_udiJsonMaxLengthOfString);
    fbJson : FFU.FB_CreateJsonFormattedString;
    fbWriteFile : FFU.FB_WriteFile;
END_VAR
```

```
//JSON Example - Implementation Part
CASE iState OF
0 : //Idle
    IF xStart AND NOT xStartOld THEN
        iState := 10 ;
        xProcessCompleted := FALSE ;
    END_IF
    xStartOld := xStart ;
    xStart := FALSE ;

10 : //Prepare data
    dtTimestamp := DWORD_TO_DT ( SysTimeRtc.SysTimeRtcGet ( udiResult )
) ; // get the RTC of the controller
    iState := 20 ;

20 : //Create String
    xErr := FALSE ;
    xErr S= NOT fbJson.New ( ) ;
    xErr S= NOT fbJson.AddNameValuePairObject ( 'Header' ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'MachineId' , 'SEMSOL-
M251MESE-F40A0B07' ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'MachineType' , 'SMR-550'
) ;
    xErr S= NOT fbJson.AddNameValuePair ( 'Timestamp' , dtTimestamp ) ;
    xErr S= NOT fbJson.AddNameValuePair ( 'MessageId' , udiMessageId ) ;
    xErr S= NOT fbJson.ObjectClose ( ) ; // close object header'
    xErr S= NOT fbJson.AddNameValuePairObject ( 'Values' ) ;
    xErr S= NOT fbJson.AddNameValuePairObject ( 'Configuration' ) ;
    xErr S= NOT fbJson.AddNameValuePairArray ( 'Devices' ) ;
    xErr S= NOT fbJson.ObjectOpen ( ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'ID' , '1' ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'Category' , 'Controller'
) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'Type' , 'TM251MESE' ) ;
    xErr S= NOT fbJson.ObjectClose ( ) ;
    xErr S= NOT fbJson.ObjectOpen ( ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'ID' , '2' ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'Category' , 'ServoDrive'
) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'Type' , 'Lexium32M' ) ;
    xErr S= NOT fbJson.ObjectClose ( ) ;
    xErr S= NOT fbJson.ArrayClose ( ) ; // close array Devices
    xErr S= NOT fbJson.ObjectClose ( ) ; // close object Configuration
    xErr S= NOT fbJson.AddNameValuePairObject ( 'Status' ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'Machinestate' ,
```

```
'Running' ) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'Errorstate' , 'NoError'
) ;
    xErr S= NOT fbJson.AddNameValuePairString ( 'Warning' , 'Filllevel
low' ) ;
    xErr S= NOT fbJson.ObjectClose ( ) ;
    xErr S= NOT fbJson.AddNameValuePairArray ( 'Processdata' ) ;
    xErr S= NOT fbJson.ObjectOpen ( ) ;
    xErr S= NOT fbJson.AddNameValuePairArray ( 'Energy' ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rTotalEnergy ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rMaxVoltage ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rLowVoltage ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rMaxCurrent ) ;
    xErr S= NOT fbJson.ArrayClose ( ) ;
    xErr S= NOT fbJson.ObjectClose ( ) ;
    xErr S= NOT fbJson.ObjectOpen ( ) ;
    xErr S= NOT fbJson.AddNameValuePairArray ( 'Temperature' ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rTemperatureSensor1 ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rTemperatureSensor2 ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rTemperatureSensor3 ) ;
    xErr S= NOT fbJson.ArrayAddValue ( rTemperatureSensor4 ) ;
    xErr S= NOT fbJson.ArrayClose ( ) ;
    xErr S= NOT fbJson.ObjectClose ( ) ;
    xErr S= NOT fbJson.ArrayClose ( ) ; // close array Processdata
    xErr S= NOT fbJson.ObjectClose ( ) ; // close object Values

    // copy the string, not mandatory for the present processing but for
presentation of the feature
    fbJson.Copy ( i_anyBuffer := sJsonString ) ;

    IF xErr OR NOT fbJson.IsJsonStringValid THEN
        iState := 100 ;
        xJsonStringInvalid := NOT fbJson.IsJsonStringValid ;
    ELSE
        iState := 30 ;
    END_IF

30 : //Start write file
    fbWriteFile (
        i_xExecute := TRUE ,
        i_sFilePath := 'MachineData.json' ,
        i_etModeFileOpen := FFU.ET_ModeFileOpen.CreatePlus ,
        i_timTimeout := ,
        i_pbyBuffer := fbJson.PointerToJsonString ,
        i_udiSize := fbJson.LengthOfJsonString ,
```

```
        q_xDone => ,
        q_xBusy => ,
        q_xError => ,
        q_etResult => ,
        q_sResultMsg => ,
        q_udiFileSize => ) ;

    iState := 40 ;

40 : // Wait for write file complete
    IF fbWriteFile.q_xDone THEN
        iState := 0 ;
        xProcessCompleted := TRUE ;
    ELSIF fbWriteFile.q_xError THEN
        iState := 100 ;
        xErrorWriteFile := TRUE ;
        etResult := fbWriteFile.q_etResult ;
        sResultMsg := fbWriteFile.q_sResultMsg ;
    END_IF

        fbWriteFile ( i_xExecute := ( iState = 40 ) ) ;

100 : //Error detected
    IF xReset THEN
        iState := 0 ;
        etResult := FFU.ET_Result.Idle ;
        sResultMsg := 'Ready for start' ;
        xJsonStringInvalid := FALSE ;
        xErrorWriteFile := FALSE ;
        xReset := FALSE ;
    END_IF
END_CASE
```

Resulting JSON File


```
{
    "Header": {
        "MachineId": "SEMSOL-M251MESE-F40A0B07",
        "MachineType": "SMR-550",
        "Timestamp": "DT#2018-02-21-17:06:39",
        "MessageId": 12587
    },
    "Values": {
        "Configuration": {
            "Devices": [{
                "ID": "1",
                "Category": "Controller",
                "Type": "TM251MESE"
            },
            {
                "ID": "2",
                "Category": "ServoDrive",
                "Type": "Lexium32M"
            }]
        },
        "Status": {
            "Machinestate": "Running",
            "Errorstate": "NoError",
            "Warning": "Fill-level low"
        },
        "Processdata": [{
            "Energy": [35.6,
            402.56,
            399.03,
            18.459]
        },
        {
            "Temperature": [21.65,
            32.47,
            36.52,
            47.35]
        }]
    }
}
```


NOTE: The original text STRING does not contain any blank spaces or line breaks. The JSON structure displayed in the figure above has been automatically created when opening the file in an appropriate text editor supporting JSON format.

Index

A

aliases, *43*

C

common inputs and outputs
  behavior of function blocks with the input i_xExecute, *21*

E

ET_CsvReadMode, *27*
  AllValues, *27*
  GetFileInformation, *27*
  OneColumn, *27*
  OneRow, *27*
  OneValue, *27*
ET_ModeFileOpen, *28*
  Append, *28*
  AppendPlus, *28*
  Create, *28*
  CreatePlus, *28*
  NotSet, *28*
ET_Result, *29*
  AdditionalContentInvalid, *30*
  AnalyzingFile, *29*
  AttributeAlreadyExistsForSelectedElement, *31*
  AttributeNotFound, *31*
  BufferFull, *30*
  BufferInvalid, *32*
  BufferSizeInvalid, *32*
  BufferTooSmall, *32*
  CheckingInputs, *29*
  ClosingFile, *29*
  DataTypeNotSupported, *31*
  ElementNotFound, *30*
  FileAlreadyExists, *30*
  FileCloseFailed, *29*
  FileFormatUtility, *29*
  FileInconsistent, *30*
  FileInvalid, *29*
  FilenameInvalid, *31*
  FilenameTooLong, *31*
  FileNotExists, *30*
  FileOpenFailed, *29*
  FilePathInvalid, *29*
  FilePathTooLong, *31*
  FileReadFailed, *30*
  FileWriteFailed, *29*
  FirstItemInvalidParentIndex, *31*
  FirstItemNoElement, *31*
  GetFileSize, *29*
  GetFileSizeFailed, *30*
  Idle, *29*
  Initializing, *29*
  InvalidInput, *31*
  MaxLengthOfJsonStringReached, *32*
  NameValuePairCannotBeAdded, *31*
  NoArrayOpen, *31*
  NoElementSelected, *31*
  NoObjectOpen, *31*
  NumOfLevelsExceeded, *32*
  NumOfParentsExceeded, *30*
  OK, *29*
  OnlineChangeDetected, *31*
  OpeningFile, *29*
  ParsingFailed, *30*

ReadingFile, *29*
ReadParameterInvalid, *30*
TableInvalid, *30*
TableReadValuesInvalid, *30*
TableTooSmall, *30*
TableWriteValuesInvalid, *30*
Timeout, *29*
UnexpectedProgramBehaviour, *32*
ValueNotFound, *30*
WriteParameterInvalid, *30*
WritingFile, *29*
WrongLayerToAppendElement, *31*
XmlItemTypeInvalid, *30*
XmlStructureInconsistent, *30*
XPathExpressionInvalid, *29*
XpathExpressionNotSupported, *31*
XpathRootElementDoesNotMatch, *31*
ET_XmlItemType, *26*
Attribute, *26*
Element, *26*
NotSet, *26*

F

FB_CreateJsonFormattedString, *130*, *130*
FB_CreateJsonFormattedString example, *146*
FB_CsvRead, *112*
FB_CsvRead considerations, *116*
FB_CsvRead example, *118*
FB_CsvRead functional description, *113*
FB_CsvRead troubleshooting, *117*
FB_CsvWrite, *120*
FB_CsvWrite considerations, *123*
FB_CsvWrite example, *125*
FB_CsvWrite functional description, *121*
FB_CsvWrite troubleshooting, *124*
FB_WriteFile, *60*, *61*, *63*
FB_WriteFile troubleshooting, *63*
FB_XmlItemsUtility, *84*, *85*
FB_XmlItemsUtility}
properties, *99*
FB_XmlRead, *68*
FB_XmlRead considerations, *72*
FB_XmlRead example, *74*
FB_XmlRead functional description, *69*
FB_XmlRead troubleshooting, *73*
FB_XmlWrite, *76*, *77*, *80*, *82*
FB_XmlWrite troubleshooting, *81*
FC_EtResultToString, *55*
FC_XmlGetElementValue, *102*
FC_XmlGetElementValue considerations, *104*
FC_XmlGetElementValue functional description, *103*
FC_XmlSetElementValue, *105*
FC_XmlSetElementValue considerations, *107*
FC_XmlSetElementValue functional description, *106*
FileFormatUtility, *19*
GCL (Global Constants List), *47*
GPL, *49*

G

GCL (Global Constants List)
FileFormatUtility, *47*
GPL
FileFormatUtility, *49*
GVL (Global Variables List), *51*

M

method
AddAttribute, *94*
AddNameValuePair, *136*, *136*
AddNameValuePairArray, *135*
AddNameValuePairBOOL, *137*
AddNameValuePairDINT, *137*
AddNameValuePairINT, *137*
AddNameValuePairNull, *139*
AddNameValuePairObject, *134*
AddNameValuePairREAL, *137*
AddNameValuePairSTRING, *137*
AddNameValuePairUDINT, *137*
AddSubElement, *96*
AppendElement, *97*
ArrayAddArray, *140*
ArrayAddBOOL, *142*
ArrayAddDINT, *142*
ArrayAddINT, *142*
ArrayAddNull, *143*
ArrayAddREAL, *142*
ArrayAddSTRING, *142*
ArrayAddUDINT, *142*
ArrayAddValue, *140*, *140*
ArrayClose, *144*
ClearSelection, *88*
Copy, *131*
GetAttributeByName, *92*
InitializeXmlItems, *95*
New, *131*
ObjectClose, *133*
ObjectOpen, *132*
RemoveAttribute, *95*
SelectElement, *86*
SelectFirstChild, *89*
SelectNext, *90*
SelectParent, *91*
SetAttributeByName, *93*

P

properties of FB_XmlItemsUtility}, *99*

S

ST_CsvFileInformation, *39*
ST_CsvReadParameter, *41*
ST_CsvTable, *37*
ST_CsvWarnValueTruncated, *40*
ST_CsvWriteParameter, *42*
ST_XmlItem, *34*
ST_XmlUserDefinedHeader, *36*

X

XmlItems, *43*

EcoStruxure Machine Expert

PackML

Library Guide

06/2019

EIO0000002809.00

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

Safety Information 5
About the Book 9
Part I General Information 15
Chapter 1 Presentation of the Library 17
General Information 18
Operation Modes 24
PackTags 26
Part II Data Unit Types 27
Chapter 2 Enumerations 29
ET_Cmd 30
ET_Modes 31
ET_Result 32
ET_States 34
Chapter 3 Structures 37
3.1 PackTags Main Structures 38
ST_Administration 39
ST_Command 41
ST_Status 42
3.2 Other Main Structures 44
ST_InitAlarm 45
ST_UnitModeDefinition 46
ST_VisInterface 48
3.3 SubStructures 53
ST_Alarm 54
ST_CountDescrip 55
ST_Descriptor 56
ST_EquipmentInterlock 57
ST_Ingredient 58
ST_Interface 59
ST_Product 60
ST_StateInfo 61
ST_Timestamp 62
Chapter 4 Aliases - DateTimeArray 63
DateTimeArray 63

Part III Global Variables 65
Chapter 5 Global Constants List 67
GCL 67
Chapter 6 Global Parameter List 69
GPL 69
Part IV Program Organization Units (POU) 73
Chapter 7 Function Blocks 75
FB_DataManagement 76
FB_ModeManager 79
FB_VisController 82
Chapter 8 Functions 85
FC_CheckCmd 86
FC_EtResultToString 88
FC_GetDateTimeAsArray 89
FC_InitStateModelChangeStates 90
FC_InitStateModelExistingStates 93
FC_SetAlarm 95
FC_SetWarning 97
Part V Visualization 99
Chapter 9 BackgroundFrames 101
FR_<BackgroundFrame> 101
Chapter 10 States Frames 103
FR_<State> 103
Chapter 11 Visualization Frames 105
General 106
FR_Alarm 107
FR_AlarmHistory 108
FR_CurrentModeAndStateTime 109
FR_DynStateModel 110
FR_ModeAndStateTime 112
FR_ProdConsumedCount 114
FR_ProdDefectiveCount 115
FR_ProdProcessedCount 116
FR_Warning 117
FR_StopReason 118
FR_StatesDisabled 119
Glossary 121
Index 123

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

BEFORE YOU BEGIN

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

⚠ WARNING

UNGUARDED EQUIPMENT

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

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

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

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

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

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

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

START-UP AND TEST

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

⚠ WARNING

EQUIPMENT OPERATION HAZARD

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

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

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

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

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

Before energizing equipment:

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

OPERATION AND ADJUSTMENTS

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

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

About the Book

At a Glance

Document Scope

This document describes the PackML library.

The library provides functions and function blocks, structures, and visualization frames as tools to help you create applications conforming to PackML and PackTags, as defined by ANSI/ISA TR88.00.02-2015.

The library is not, and cannot be, a full implementation of the ANSI/ISA TR88.00.02-2015 standard, as you are responsible for the application code to support your machine or process. Instead, the library is a tool to help you to deploy the PackML standard for machine application.

Validity Note

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

The technical characteristics of the devices described in the present document also appear online. To access the information online:

Step	Action
1	Go to the Schneider Electric home page *www.schneider-electric.com*.
2	In the **Search** box type the reference of a product or the name of a product range. ● Do not include blank spaces in the reference or product range. ● To get information on grouping similar modules, use asterisks (*).
3	If you entered a reference, go to the **Product Datasheets** search results and click on the reference that interests you. If you entered the name of a product range, go to the **Product Ranges** search results and click on the product range that interests you.
4	If more than one reference appears in the **Products** search results, click on the reference that interests you.
5	Depending on the size of your screen, you may need to scroll down to see the data sheet.
6	To save or print a data sheet as a .pdf file, click **Download XXX product datasheet**.

The characteristics that are presented in the present document should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the document and online information, use the online information as your reference.

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.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

⚠ 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.

Incomplete file transfers, such as data files, application files and/or firmware files, may have serious consequences for your machine or controller. If you remove power, or if there is a power outage or communication interruption during a file transfer, your machine may become inoperative, or your application may attempt to operate on a corrupted data file. If an interruption occurs, reattempt the transfer. Be sure to include in your risk analysis the impact of corrupted data files.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION, DATA LOSS, OR FILE CORRUPTION

- Do not interrupt an ongoing data transfer.
- If the transfer is interrupted for any reason, re-initiate the transfer.
- Do not place your machine into service until the file transfer has completed successfully, unless you have accounted for corrupted files in your risk analysis and have taken appropriate steps to prevent any potentially serious consequences due to unsuccessful file transfers.

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

The POUs provided with this library use variables of type POINTER TO internally. These pointers are assigned only on start of execution of the respective POU. This is, the pointers are not reassigned while the function block indicates `Busy`.

⚠ CAUTION

INVALID POINTER

Do not use the “Online Change” command or the “Log in with online change” option as long as one of the function blocks of this library indicates `Busy` in your running application.

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

Related Documents

Document title	Reference
EcoStruxure Machine Expert Functions and Libraries User Guide	*EIO0000002829 (ENG)*; *EIO0000002830 (FRE)*; *EIO0000002831 (GER)*; *EIO0000002832 (ITA)*; *EIO0000002833 (SPA)*; *EIO0000002834 (CHS)*;
EcoStruxure Machine Expert Programming Guide	*EIO0000002854 (ENG)*; *EIO0000002855 (FRE)*; *EIO0000002856 (GER)*; *EIO0000002857 (ITA)*; *EIO0000002858 (SPA)*; *EIO0000002859 (CHS)*;

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2015	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.

Standard	Description
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2016	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

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

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

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

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

Part I
General Information

Chapter 1
Presentation of the Library

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
General Information	18
Operation Modes	24
PackTags	26

General Information

Library Overview

The library provides components to facilitate the implementation of an application based on the PackML standard. The names and structure of the data types provided in this library correspond to the definitions in ANSI/ISA TR88.00.02-2015.

Characteristics of the Library

Characteristic	Value
Library title	PackML
Company	Schneider Electric
Category	**Application → Solution → Packaging**
Component	**Packaging** (**Application Libraries and Templates → Packaging**)
Default namespace	**PackML**
Language model attribute	qualified-access-only *(see EcoStruxure Machine Expert, Functions and Libraries User Guide)*
Forward compatible library	Yes (FCL *(see EcoStruxure Machine Expert, Functions and Libraries User Guide)*)

NOTE: For this library, qualified-access-only is set. The POUs, data structures, enumerations, and constants must be accessed using the namespace of the library. The default namespace of the library is **PackML**.

Example Project

In conjunction with the library, the example project PackMLExample.project is provided. The example project demonstrates how to implement the components from the PackML library.

Your specific application requirements may be different from those assumed for any related examples, templates or architectures described herein. In that case, you will have to adapt the information provided in this and other related documents to your particular needs. To do so, you will need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any information specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to your adaptation.

⚠ WARNING
REGULATORY INCOMPATIBILITY Be sure 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 example project is installed on your PC along with the programming software. To open the project example, proceed as follows:

Step	Action	Comment
1	In the EcoStruxure Machine Expert Logic Builder, execute the command **New Project**.	–
2	In the **New Project** dialog box, select **From Example** from the **Project type** list.	–
3	On the right-hand side of the **New Project** dialog box, select **All** from the **Controller** list.	**Result**: Available examples are listed in the **Matching Examples** text box.
4	Select your example from the **Matching Examples** list.	–
5	Enter a name for the new project, and select the file location.	–
6	Click the **OK** button.	**Result**: A new project is created based on the selected example.

Overview of the POUs

The library contains the following function blocks and functions:

Function block / function	Use
`FB_DataManagement` *(see page 76)*	Records time and production information.
`FB_ModeManager` *(see page 79)*	Verifies whether a change of operation mode is possible within the active state and performs consistency verifications on the definition of operation modes.
`FB_VisController` *(see page 82)*	Processes data provided by the administration tags and status tags and provides these to the visualization frames via the visualization interface input/output `iq_stVisInterface` of type `ST_VisInterface`.
`FC_CheckCmd` *(see page 86)*	Verifies whether the given command can be accepted in the present state (according to the given operation mode definition). If so, it also returns the target state of the transition triggered by that command.

Function block / function	Use
`FC_InitStateModelChangeStates` *(see page 90)*	Initializes in which states a change of operation mode is permitted for an operation mode.
`FC_InitStateModelExistingStates` *(see page 93)*	Initializes which states exist for an operation mode.
`FC_SetAlarm` *(see page 95)*	Writes alarm messages into the administration tags `Admin.Alarm[#]` and `Admin.AlarmHistory[#]`.
`FC_SetWarning` *(see page 97)*	Writes advisory messages into the administration tag `Admin.Warning[#]`.
`FC_GetDateTimeAsArray` *(see page 89)*	Obtains the real-time clock (RTC) of the controller and converts it to the data type `DateTimeArray` *(see page 63)*.

Overview of the Enumerations

Enumeration	Use
`ET_Cmd` *(see page 30)*	Contains the commands to change the state within a given operation mode. Commands are used to transition from a **Wait state** to other states (for example, the `Start` command). Additionally, the commands `Stop` and `Abort` can be issued during any state (refer to the State Model Diagram *(see page 25)*).
`ET_Modes` *(see page 31)*	Contains predefined operation modes (for example, `Producing`).
`ET_Result` *(see page 32)*	Contains the possible values that indicate the result of operations executed by the function block.
`ET_States` *(see page 34)*	Contains the possible states of which an operation mode can be comprised.

Overview of the Structures

Structure	Use
`ST_Administration` *(see page 39)*	Provides the variables/tags which contain OEE (Overall Equipment Effectiveness) data (for example, downtime, reasons for downtime, etc). Furthermore production information can be provided by this variable (for example, alarm and state information).
`ST_Command` *(see page 41)*	Provides the variables which mainly represent the control interface. By using these tags, operation modes can be selected, state transitions can be forced, and recipes as well as formats can be transmitted.

Structure	Use
ST_InitAlarm *(see page 45)*	Provides information to set an alarm message in the function FC_SetAlarm.
ST_UnitModeDefinition *(see page 46)*	Provides information to initialize the operation modes in the function block FB_ModeManager.
ST_Status *(see page 42)*	Provides the variables which contain status information (such as which operation mode is active, which state is active, etc).
ST_VisInterface *(see page 48)*	Structure for visualization.
ST_Alarm *(see page 54)*	Provides the machine diagnostic message number.
ST_CountDescrip *(see page 55)*	Provides the ID and parameter of count products.
ST_Descriptor *(see page 56)*	Provides information of the parameter (id, name, unit, and value).
ST_Ingredient *(see page 58)*	Provides ID and parameter of ingredient
ST_Interface *(see page 59)*	Provides information about the downstream (subsequent) PackML machine.
ST_Product *(see page 60)*	Provides information about a product.
ST_StateInfo *(see page 61)*	Provides the information whether the state exists and whether a change of operation mode is possible. The information is used to initialize the operation mode manager.
ST_Timestamp *(see page 62)*	Provides the date and time.

Overview of the Frames

Background frames

The FR_<BackgroundFrame> *(see page 101)* visualization frames (for example, FR_Control1024x768) are frames in the background to group the other frames, and to realize control and navigation functionalities.

Frame
FR_Control1024x768
FR_Nav1024x768
FR_NavControl1024x768
FR_Control800x600
FR_Nav800x600
FR_NavControl800x600

State frames

The FR_<State> *(see page 103)* visualization frames (for example, FR_Stopped) are frames for each state of a model.

The frames are used by the FR_DynStateModel *(see page 110)* to display the single states of the state model for operation modes.

Frame
FR_Aborted
FR_Aborting
FR_Clearing
FR_Complete
FR_Completing
FR_Execute
FR_Held
FR_Holding
FR_Idle
FR_Resetting
FR_Starting
FR_Stopped
FR_Stopping
FR_Suspended
FR_Suspending
FR_UnHolding
FR_UnSuspending

Visualization frames

In addition to the individual tags, functions, and function blocks, the library contains some visualization frames, which dynamically generate an appropriate state model for the operation mode during online operation or display OEE (Overall Equipment Effectiveness) data.

Frame	Use
FR_Alarm *(see page 107)*	Displays an alarm list of the active alarms.
FR_AlarmHistory *(see page 108)*	Displays the alarm history.
FR_CurrentModeAndStateTime *(see page 109)*	Displays the time the machine has been running in the present operation mode/state.
FR_DynStateModel *(see page 110)*	Dynamically displays the visualization of the state model during online operation.
FR_ModeAndStateTime *(see page 112)*	Displays the operation time of the operation modes and their respective states.
FR_ProdConsumedCount *(see page 114)*	Displays the product counters of the consumables.

Frame	Use
FR_ProdDefectiveCount *(see page 115)*	Displays the product counters of the goods detected as not acceptable.
FR_ProdProcessedCount *(see page 116)*	Displays the product counters of the produced goods.
FR_Warning *(see page 117)*	Displays a list of the active advisories.
FR_StopReason *(see page 118)*	Displays the PackTag Admin.StopReason.
FR_StatesDisabled *(see page 119)*	Monitors and controls the PackTag Admin.StatesDisabled.

Operation Modes

Overview

PackML guidelines define standardized state model to demonstrate the fundamental behavior of a packaging machine in a standardized manner.

According to the PackML guidelines each packaging machine may support many operation modes.

Each operation mode can be comprised of up to 17 states (**Aborted**, **Aborting**, **Clearing**, **Complete**, **Completing**, **Execute**, **Held**, **Holding**, **Idle**, **Resetting**, **Starting**, **Stopped**, **Stopping**, **Suspended**, **Suspending**, **Un-Holding** and **Un-Suspending**).

To move from one state to another, there are transitions between states. Each transition is triggered by either a state transition command or the SC (State Complete) condition.

The implementation of the PackML library focuses on the unit control mode as described in ANSI/ISA TR88.00.02-2015 standard. In all cases, when the present document refers to operation mode, it is a specific reference to unit control mode.

Using State Models

State models in the PackML library are based on the state model with 17 states, as defined in ANSI/ISA TR88.00.02-2015, refer to the diagram thereafter. A state model must contain these states, or a subset of these states.

A state model is defined by specifying which of these states exist for the operation mode. Transitions between states are created automatically and implicitly based on which states are selected.

`ST_UnitModeDefinition` is the structure representing the state model for an operation mode.

`FC_InitStateModelExistingStates` initializes which states exist.

`FC_InitStateModelChangeStates` initializes which states allow a change of operation mode.

`FB_ModeManager` verifies the state model for consistency and verifies whether a change of operation mode is possible and the target state in that operation mode.

`FC_CheckCmd` verifies whether a state transition command is valid and returns the resulting target state.

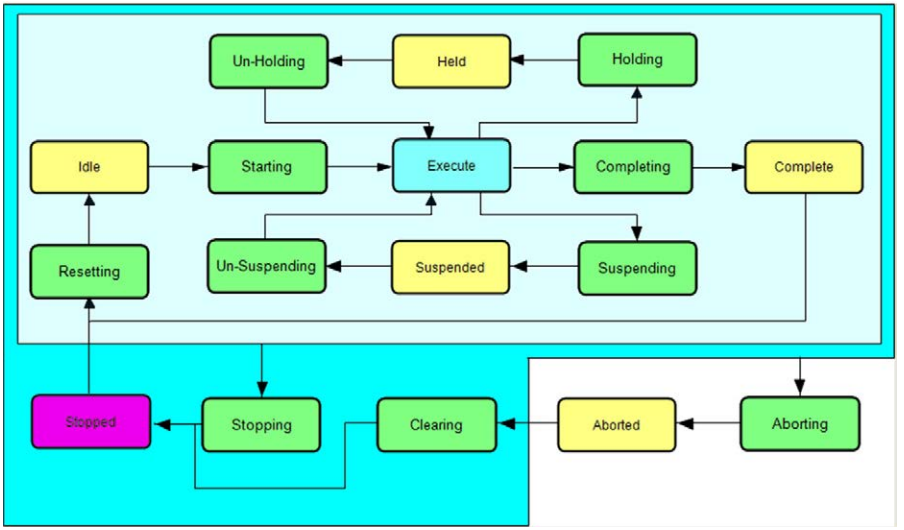
Visualization frames and `FB_VisController` are used to visualize the state model and provide a commissioning interface for it.

You are responsible for implementing the correct flow in the state model according to the transitions defined in the state model, as well as for implementing changes of operation mode. POUs provided in the library can help to implement this logic (`FB_ModeManager` and `FC_CheckCmd`).

You are responsible for implementing the tracking of the state and operation mode, as well as for implementing a machine behavior that corresponds to the states, the transitions, and the operation modes.

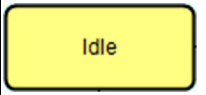
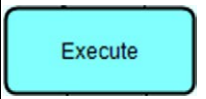
State Model Diagram

The following diagram illustrates the state model for operation modes with the 17 states present:



Type of States

Within the state model, three types of states are differentiated:

Type of state	Description from ANSI/ISA TR88.00.02-2015	Example
Wait state	A state used to identify that a machine has achieved a defined set of conditions. In such a state, the machine is maintaining a status until transitioning to an acting state.	
Acting state	A state which represents some processing activity. It implies the single or repeated execution of processing steps in a logical order, for a finite time or until a specific condition has been reached.	 

Boundaries

The following list describes the boundaries which define operation modes based on the state model:

- Maximum number of states: 17
  The states for the operation mode can be selected from those pre-defined in the state model. Each state can appear only once and whose position is specified by the state model. You cannot add additional states, nor change the order of the states.
- State model is unidirectional.
- The state following a **Wait state** must be an **Acting state**.
- Logic can be executed in each state.

NOTE: More rules apply to build operation modes with the PackML library. Refer to `FB_ModeManager` *(see page 79)*.

PackTags

Overview

PackTags provide a uniform set of naming conventions for data elements used within the procedural elements of the base state model. They are named data elements used for open architecture and interoperable data exchange in automated machinery.

The library provides the PackTags, as defined in ANSI/ISA TR88.00.02-2015.

Variables

The variables are grouped into three types:

PackTags type	Description	In the PackML Library
command	These variables mainly represent the control interface. By using these tags, operation modes can be selected, state transitions can be forced, and recipes as well as formats can be transmitted.	`ST_Command`
status	These variables contain status information (such as which operation mode is active, which state is active, and so on).	`ST_Status`
administration	These variables contain important OEE data (for example, downtime, reasons for downtime, and so on). Furthermore production information.	`ST_Administration`

The individual variables from the three types (command, status, administration) can be provided on the controller within the structures.

Part II
Data Unit Types

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
2	Enumerations	29
3	Structures	37
4	Aliases - `DateTimeArray`	63

Chapter 2
Enumerations

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
ET_Cmd	30
ET_Modes	31
ET_Result	32
ET_States	34

ET_Cmd

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

The enumeration includes the state commands used to proceed a state transition. A **Wait state** *(see page 24)* can only be exited with a state transition command (for example `Start`).

Enumeration Elements

Name	Data type	Value	Description
`Undefined`	DINT	0	Commands to transition from one state to another are described in `FC_CheckCmd` *(see page 86)*.
`Reset`	DINT	1	
`Start`	DINT	2	
`Stop`	DINT	3	
`Hold`	DINT	4	
`UnHold`	DINT	5	
`Suspend`	DINT	6	
`UnSuspend`	DINT	7	
`Abort`	DINT	8	
`Clear`	DINT	9	

Used By

The values of the enumeration can be used as an input value for `i_diCmd` of `FC_CheckCmd` or to fill in the value for the variable `CntrlCmd` of `ST_Command`.

ET_Modes

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

The enumeration includes predefined operation modes (for example, production). An operation mode is an ordered subset of states, commands, and transitions which can be initialized via the structure `ST_UnitModeDefinition`.

Enumeration Elements

Name	Data type	Value	Description
`Undefined`	DINT	0	The operation mode is undefined or invalid.
`Producing` (1)	DINT	1	Operation mode for production.
`Maintenance`	DINT	2	Operation mode for maintenance.
`Manual`	DINT	3	Manual operation mode

(1) With the 2015 revision of the ANSI/ISA TR88.00.02-2015, the control mode `Producing` was changed to `Production`. For reasons of compatibility with the previous version of the PackML library, the name `Producing` has been maintained.

Used By

The values of the enumeration can be used to fill values of the variables `UnitModeCurrent` of `ST_Status` and `UnitMode` of `ST_Command`.

ET_Result

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

The enumeration includes the possible values that indicate the result of operations executed by the function block.

Enumeration Elements

Name	Data type	Value	Description
If q_xError of a function block *(see page 75)* is FALSE, one of the following status messages is shown.			
Ok	DINT	0	The POU is OK, but not running.
Disabled	DINT	1	The POU is disabled.
Initializing	DINT	2	The POU is being initialized.
Running	DINT	4	The POU is being executed.
ResettingTimerAndCounter	DINT	5	The POU is resetting the timer and counter.
If q_xError of a function block *(see page 75)* is TRUE, one of the following error messages is shown.			
UnitModeCurrentRange	DINT	11	The value of the input i_diUnitModeCurrent is out of range. The value of operation mode must be in the range 0 and Gc_uiMaxNumberOfModes.
StateCurrentRange	DINT	12	The value of the input i_diStateCurrent is out of range. The value of state must be in the range 0 and Gc_uiMaxNumberOfStates.
NumberOfModesRange	DINT	13	Number of operation modes is out of range. The number of operation modes must be in the range 0 and Gc_uiMaxNumberOfModes.
PointerInitModeInvalid	DINT	14	The initial operation mode pointer i_pstInitMode is invalid.
PointerAdminInvalid	DINT	15	The admin pointer i_pstAdmin is invalid.
PointerStatusInvalid	DINT	16	The status pointer i_pstStatus is invalid.

Name	Data type	Value	Description
UnitModeRange	DINT	17	The operation mode out of range. Range should be: 0 < i_diUnitMode <= diNumberOfModes
InvalidStateModelNoExecuteState	DINT	20	The state model of the operation mode is invalid: there is no **Execute** state.
InvalidStateModelNoStoppedState	DINT	21	The state model of the operation mode is invalid: there is no **Stopped** state.
InvalidStateModelNoHeldState	DINT	22	The state model of the operation mode is invalid: a **Held** state must exist if a **Holding** or an **Un-Holding** state is present.
InvalidStateModelNoCompleteState	DINT	23	The state model of the operation mode is invalid: if **Completing** state exists, a **Complete** state must be present.
InvalidStateModelNoSuspendedState	DINT	24	The state model of the operation mode is invalid: a **Suspended** state must exist if a **Suspending** or an **Un-Suspending** state is present.
InvalidStateModelNoAbortedState	DINT	25	The state model of the operation mode is invalid: an **Aborted** state must exist if an **Aborting** or a **Clearing** state is present.
InvalidStateModelNoIdleState	DINT	26	The state model of the operation mode is invalid: the **Resetting** state can be used only in connection with the **Idle** state.
InvalidStateModelNoResettingState	DINT	27	The state model of the operation mode is invalid: the **Idle** state can be used only in connection with the **Resetting** state.
ModeChangeRequestRejected	DINT	28	The request to change the operation mode is rejected.
UnknownResult	DINT	30	The value of the input i_etResult is invalid.

Used By

- FB_DataManagement
- FB_ModeManager
- FB_VisController
- FC_EtResultToString

ET_States

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

This enumeration represents the list of possible machine states as defined in ANSI/ISA TR88.00.02-2015.

Enumeration Elements

NOTE: For a detailed description of each of the individual states, refer to the 2015 revision of the standard ANSI/ISA TR88.00.02-2015.

Name	Data type	Value	Description
Undefined	DINT	0	No state is defined.
Clearing	DINT	1	**Acting state Clearing**
Stopped	DINT	2	**Wait state Stopped**
Starting	DINT	3	**Acting state Starting**
Idle	DINT	4	**Wait state Idle**
Suspended	DINT	5	**Wait state Suspended**
Execute	DINT	6	**Acting state Execute**
Stopping	DINT	7	**Acting state Stopping**
Aborting	DINT	8	**Acting state Aborting**
Aborted	DINT	9	**Wait state Aborted**
Holding	DINT	10	**Acting state Holding**
Held	DINT	11	**Wait state Held**
UnHolding	DINT	12	**Acting state Un-Holding**
Suspending	DINT	13	**Acting state Suspending**
UnSuspending	DINT	14	**Acting state Un-Suspending**
Resetting	DINT	15	**Acting state Resetting**
Completing	DINT	16	**Acting state Completing**
Complete	DINT	17	**Wait state Complete**
NOTE: The contents of any given state is subject to your application.			

Used By

- FB_ModeManager
- FB_VisController
- i_diStateCurrent of FC_CheckCmd
- StateCurrent of ST_Status
- StateRequested of ST_Status

Chapter 3
Structures

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
3.1	PackTags Main Structures	38
3.2	Other Main Structures	44
3.3	SubStructures	53

Section 3.1
PackTags Main Structures

What Is in This Section?

This section contains the following topics:

Topic	Page
ST_Administration	39
ST_Command	41
ST_Status	42

ST_Administration

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_Administration represents the list of PackTags of type Administration as defined in ANSI/ISA TR88.00.02-2015.

Administration tags are used to describe the quality and alarm information of the unit machine.

Structure Elements

For the GPL data types, refer to Global Parameters *(see page 69)*.

Variable	Data type	Description
Parameter	ARRAY [1..GPL.Gc_iNumOfAdminParameter] OF ST_Descriptor	Array of ST_Descriptor *(see page 56)*
Alarm	ARRAY [1..GPL.Gc_uiNumberOfAlarms] OF ST_Alarm	Array of ST_Alarm *(see page 54)*
AlarmExtent	DINT	Size of alarm array
AlarmHistory	ARRAY [1..GPL.Gc_uiNumberOfAlarmHistory] OF ST_Alarm	Array of given size for machine detected error code and messaging *(see page 54)*
AlarmHistoryExtent	DINT	Size of alarm history array
ModeCurrentTime	ARRAY [0..GPL.Gc_uiMaxNumberOfModes] OF UDINT	Time spent in operation mode
ModeCumulativeTime	ARRAY [0..GPL.Gc_uiMaxNumberOfModes] OF UDINT	Cumulative time in each operation mode
StateCurrentTime	ARRAY [0..GPL.Gc_uiMaxNumberOfModes,0..GVL.Gc_uiMaxNumberOfStates] OF UDINT	Value of last state timer in each operation mode/state
(1) Variable has been replaced by PLCDateTime in ANSI/ISA TR88.00.02-2015 but remains in this library for compatibility reasons.		

Variable	Data type	Description
StateCumulativeTime	ARRAY [0..GPL.Gc_uiMaxNumberOfModes,0..GVL.Gc_uiMaxNumberOfStates] OF UDINT	Cumulative time in each operation mode/state
ProdConsumedCount	ARRAY [1..GPL.Gc_uiNumberOfConsumedProducts] OF ST_CountDescrip	Array of ST_CountDescrip *(see page 55)* (of products consumed in the production machine)
ProdProcessedCount	ARRAY [1..GPL.Gc_uiNumberOfProcessedProducts] OF ST_CountDescrip	Array of ST_CountDescrip (of products processed by the production machine)
ProdDefectiveCount	ARRAY [1..GPL.Gc_uiNumberOfDefectiveProducts] OF ST_CountDescrip	Array of ST_CountDescrip (of products that are marked as not acceptable in the production machine)
AccTimeSinceReset	DINT	Accumulative time since last Reset command
MachDesignSpeed	REAL	Machine design speed
PACDateTime(1)	ST_Timestamp	Time stamp (ISO date and time data type, ISO 8601:1988 format) *(see page 62)*
StopReason	ST_Alarm *(see page 54)*	Information about the first event captured during an abort, held, suspended, or stop event
Warning	ARRAY [1..GPL.Gc_uiMaxNumberOfWarnings] OF ST_Alarm	Array contains the active and/or unacknowledged machine advisories
WarningExtent	DINT	Size of advisory array.
StatesDisabled	DINT	States 1...17 of the PackML State Model can be disabled by turning on the corresponding bit
PLCDateTime	DateTimeArray *(see page 63)*	Present date and time of the programmable logic controller

(1) Variable has been replaced by PLCDateTime in ANSI/ISA TR88.00.02-2015 but remains in this library for compatibility reasons.

Used By

- FB_DataManagement
- FB_VisController

ST_Command

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_Command represents the list of PackTags of type Command as defined in ANSI/ISA TR88.00.02-2015.

Command tags are used to control the operation of the unit machine.

Structure Elements

For the GPL data types, refer to Global Parameters *(see page 69)*.

Variable	Data type	Description
UnitMode	DINT	The target operation mode (see ET_Modes *(see page 31)*)
UnitModeChangeRequest	BOOL	TRUE = request to change the machine operation mode
MachSpeed	REAL	Machine present speed
MaterialInterlocks	ARRAY [0..31] OF BOOL	Materials ready
CntrlCmd	DINT	The value of the command state
CmdChangeRequest	BOOL	TRUE = request to change the machine state
RemoteInterface	ARRAY [1..GPL.Gc_uiNumberOfInterface] OF ST_Interface *(see page 59)*	Used for coordinating upstream or downstream machines in a cell of multiple unit machines
Parameter	ARRAY [1.. GPL.Gc_uiNumberOfParameter] OF ST_Descriptor	Array of ST_Descriptor *(see page 56)*
Product	ARRAY [1.. GPL.Gc_uiNumberOfProducts] OF ST_Product	Array of ST_Product *(see page 60)*

ST_Status

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_Status represents the list of PackTags of type Status as defined in ANSI/ISA TR88.00.02-2015.

Status tags are used to describe the operation of the unit machine.

Structure Elements

For the GPL data types, refer to Global Parameters *(see page 69)*.

Variable	Data type	Description
UnitModeCurrent	DINT	Operation mode in use
UnitModeRequested	BOOL	Requested change of operation mode
UnitModeChangeInProcess	BOOL	Requested change of operation mode in process
StateCurrent	DINT	Machine present state
StateRequested	DINT	Requested state / target state
StateChangeInProgress	BOOL	Requested state transition in process
MachSpeed	REAL	PackML machine speed setpoint
CurMachSpeed	REAL	PackML machine present speed
MaterialInterlock	ARRAY [0..31] OF BOOL;	Array of ready materials
RemoteInterface	ARRAY [1.. GPL.Gc_uiNumberOfInterface] OF ST_Interface *(see page 59)*	Used for coordinating upstream or downstream machines in a cell of multiple unit machines
Parameter	ARRAY [1.. GPL.Gc_uiNumberOfParameter] OF ST_Descriptor	Machine parameter array of ST_Descriptor *(see page 56)*
Product	ARRAY [1.. GPL.Gc_uiNumberOfProducts] OF ST_Product	Array of ST_Product *(see page 60)*.
EquipmentInterlock	ST_EquipmentInterlock	Refer to ST_EquipmentInterlock *(see page 57)*.

Used By

- FB_VisController

Section 3.2
Other Main Structures

What Is in This Section?

This section contains the following topics:

Topic	Page
ST_InitAlarm	45
ST_UnitModeDefinition	46
ST_VisInterface	48

ST_InitAlarm

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_InitAlarm contains the information about the alarm or advisory which is managed using the functions FC_SetAlarm *(see page 95)* or FC_SetWarning *(see page 97)*.

Structure Elements

Variable	Data type	Description
diID	DINT	Event message ID
diValue	DINT	Event message value
sMessage	STRING[GPL.Gc_uiMaxLengthOfAlarmMessage]	Event message
xIsInList	BOOL	Event is in list
xIsAck	BOOL	Event is acknowledged
Category	DINT	Category of the event

Used By

- FC_SetAlarm
- FC_SetWarning

ST_UnitModeDefinition

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_UnitModeDefinition contains the information to initialize the operation modes in the function block FB_ModeManager *(see page 79)*.

Structure Elements

Variable	Data type	Description
xModeExistent	BOOL	TRUE: operation mode is defined.
stClearing	ST_StateInfo *(see page 61)*	Indicates whether the state stClearing is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stStopped	ST_StateInfo	Indicates whether the state stStopped is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stStarting	ST_StateInfo	Indicates whether the state stStarting is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stIdle	ST_StateInfo	Indicates whether the state stIdle is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stResetting	ST_StateInfo	Indicates whether the state stResetting is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stExecute	ST_StateInfo	Indicates whether the state stExecute is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stStopping	ST_StateInfo	Indicates whether the state stStopping is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stAborting	ST_StateInfo	Indicates whether the state stAborting is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.

Variable	Data type	Description
stAborted	ST_StateInfo	Indicates whether the state stAborted is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stHolding	ST_StateInfo	Indicates whether the state stHolding is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stHeld	ST_StateInfo	Indicates whether the state stHeld is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stUnHolding	ST_StateInfo	Indicates whether the state stUnHolding is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stSuspended	ST_StateInfo	Indicates whether the state stSuspended is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stUnSuspending	ST_StateInfo	Indicates whether the state stUnSuspending is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stSuspending	ST_StateInfo	Indicates whether the state stSuspending is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stCompleting	ST_StateInfo	Indicates whether the state stCompleting is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stComplete	ST_StateInfo	Indicates whether the state stComplete is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.

Used By

- FB_ModeManager
- FB_VisController
- FC_InitStateModelChangeStates

ST_VisInterface

Overview

Type:	Data structure
Available as of:	V1.0.1.0

Description

The structure ST_VisInterface forms the interface to the visualization screens provided with the PackML library.

Structure Elements

Variable	Data type	Description
vq_xBlink	BOOL	Variable used to flash the present **Acting state**
vq_diNumberOfModes	DINT	Maximum number of operation modes
vq_diUnitModeCurrent	DINT	Operation mode in use
vq_diStateCurrent	DINT	Present state
vq_diModeCurrentTimeActive	DINT	Present time in active operation mode
vq_diModeCumulativeTimeActive	DINT	Cumulative time in active operation mode
vq_diStateCurrentTimeActive	DINT	Present time in active state
vq_diStateCumulativetimeActive	DINT	Cumulative time in active state
vq_astAlarm	ARRAY [0..10] OF ST_Alarm	Array of given size for active machine error code and messaging
vq_astAlarmHistory	ARRAY [0..10] OF ST_Alarm	Array of given size for machine error code and messaging
vq_diProdConsumedId	DINT	ID of consumed product
vq_sProdConsumedName	STRING [GPL.Gc_uiMaxLengthOfParameterName]	Name of consumed product
vq_sProdConsumedUnit	STRING [5]	Unit of consumed product
vq_diProdConsumedCount	DINT	Count of consumed products
vq_diProdConsumedAccCount	DINT	Accumulative count of consumed products
vq_diProdProcessedId	DINT	ID of processed product

Variable	Data type	Description
vq_sProdProcessedName	STRING [GPL.Gc_uiMaxLengthOfParameterName]	Name of processed product
vq_sProdProcessedUnit	STRING [80]	Unit of processed product
vq_diProdProcessedCount	DINT	Count of processed products
vq_diProdProcessedAccCount	DINT	Accumulative count of processed products
vq_diProdDefectiveId	DINT	ID of not acceptable product
vq_sProdDefectiveName	STRING [GPL.Gc_uiMaxLengthOfParameterName]	Name of not acceptable product
vq_sProdDefectiveUnit	STRING [5]	Unit of not acceptable product
vq_diProdDefectiveCount	DINT	Count of not acceptable products
vq_diProdDefectiveAccCount	DINT	Accumulative count of not acceptable products
vq_diModeCurrentTime	DINT	Present time in each operation mode
vq_diModeCumulativeTime	DINT	Cumulative time in each operation mode
vq_diStateCurrentTime	DINT	Present time in each state
vq_diStateCumulativetime	DINT	Cumulative time in each state
vq_diTimeSinceLastReset	DINT	Time since last Reset command
vq_diStateCurrentTimeUndefined	DINT	Present time in undefined state
vq_diStateCurrentTimeClearing	DINT	Present time in **Clearing** state
vq_diStateCurrentTimeStopped	DINT	Present time in **Stopped** state
vq_diStateCurrentTimeStarting	DINT	Present time in **Starting** state
vq_diStateCurrentTimeIdle	DINT	Present time in **Idle** state
vq_diStateCurrentTimeSuspended	DINT	Present time in **Suspended** state
vq_diStateCurrentTimeExecute	DINT	Present time in **Execute** state
vq_diStateCurrentTimeStopping	DINT	Present time in **Stopping** state
vq_diStateCurrentTimeAborting	DINT	Present time in **Aborting** state
vq_diStateCurrentTimeAborted	DINT	Present time in **Aborted** state
vq_diStateCurrentTimeHolding	DINT	Present time in **Holding** state
vq_diStateCurrentTimeHeld	DINT	Present time in **Held** state
vq_diStateCurrentTimeUnholding	DINT	Present time in **Un-Holding** state
vq_diStateCurrentTimeSuspending	DINT	Present time in **Suspending** state

Variable	Data type	Description
vq_sProdProcessedName	STRING [GPL.Gc_uiMaxLengthOfParameterName]	Name of processed product
vq_sProdProcessedUnit	STRING [80]	Unit of processed product
vq_diProdProcessedCount	DINT	Count of processed products
vq_diProdProcessedAccCount	DINT	Accumulative count of processed products
vq_diProdDefectiveId	DINT	ID of not acceptable product
vq_sProdDefectiveName	STRING [GPL.Gc_uiMaxLengthOfParameterName]	Name of not acceptable product
vq_sProdDefectiveUnit	STRING [5]	Unit of not acceptable product
vq_diProdDefectiveCount	DINT	Count of not acceptable products
vq_diProdDefectiveAccCount	DINT	Accumulative count of not acceptable products
vq_diModeCurrentTime	DINT	Present time in each operation mode
vq_diModeCumulativeTime	DINT	Cumulative time in each operation mode
vq_diStateCurrentTime	DINT	Present time in each state
vq_diStateCumulativetime	DINT	Cumulative time in each state
vq_diTimeSinceLastReset	DINT	Time since last `Reset` command
vq_diStateCurrentTimeUndefined	DINT	Present time in undefined state
vq_diStateCurrentTimeClearing	DINT	Present time in **Clearing** state
vq_diStateCurrentTimeStopped	DINT	Present time in **Stopped** state
vq_diStateCurrentTimeStarting	DINT	Present time in **Starting** state
vq_diStateCurrentTimeIdle	DINT	Present time in **Idle** state
vq_diStateCurrentTimeSuspended	DINT	Present time in **Suspended** state
vq_diStateCurrentTimeExecute	DINT	Present time in **Execute** state
vq_diStateCurrentTimeStopping	DINT	Present time in **Stopping** state
vq_diStateCurrentTimeAborting	DINT	Present time in **Aborting** state
vq_diStateCurrentTimeAborted	DINT	Present time in **Aborted** state
vq_diStateCurrentTimeHolding	DINT	Present time in **Holding** state
vq_diStateCurrentTimeHeld	DINT	Present time in **Held** state
vq_diStateCurrentTimeUnholding	DINT	Present time in **Un-Holding** state
vq_diStateCurrentTimeSuspending	DINT	Present time in **Suspending** state

Variable	Data type	Description
vq_diStateCurrentTimeUnsuspending	DINT	Present time in **Un-Suspending** state
vq_diStateCurrentTimeResetting	DINT	Present time in **Resetting** state
vq_diStateCurrentTimeCompleting	DINT	Present time in **Completing** state
vq_diStateCurrentTimeComplete	DINT	Present time in **Complete** state
vq_diStateCumulativeTimeUndefined	DINT	Cumulative time in undefined state
vq_diStateCumulativeTimeClearing	DINT	Cumulative time in **Clearing** state
vq_diStateCumulativeTimeStopped	DINT	Cumulative time in **Stopped** state
vq_diStateCumulativeTimeStarting	DINT	Cumulative time in **Starting** state
vq_diStateCumulativeTimeIdle	DINT	Cumulative time in **Idle** state
vq_diStateCumulativeTimeSuspended	DINT	Cumulative time in **Suspended** state
vq_diStateCumulativeTimeExecute	DINT	Cumulative time in **Execute** state
vq_diStateCumulativeTimeStopping	DINT	Cumulative time in **Stopping** state
vq_diStateCumulativeTimeAborting	DINT	Cumulative time in **Aborting** state
vq_diStateCumulativeTimeAborted	DINT	Cumulative time in **Aborted** state
vq_diStateCumulativeTimeHolding	DINT	Cumulative time in **Holding** state
vq_diStateCumulativeTimeHeld	DINT	Cumulative time in **Held** state
vq_diStateCumulativeTimeUnholding	DINT	Cumulative time in **Un-Holding** state
vq_diStateCumulativeTimeSuspending	DINT	Cumulative time in **Suspending** state
vq_diStateCumulativeTimeUnsuspending	DINT	Cumulative time in **Un-Suspending** state
vq_diStateCumulativeTimeResetting	DINT	Cumulative time in **Resetting** state
vq_diStateCumulativeTimeCompleting	DINT	Cumulative time in **Completing** state
vq_diStateCumulativeTimeComplete	DINT	Cumulative time in **Complete** state
vq_xStateClearingExistent	BOOL	TRUE: **Clearing** state exists
vq_xStateStoppedExistent	BOOL	TRUE: **Stopped** state exists
vq_xStateStartingExistent	BOOL	TRUE: **Starting** state exists
vq_xStateIdleExistent	BOOL	TRUE: **Idle** state exists
vq_xStateSuspendedExistent	BOOL	TRUE: **Suspended** state exists
vq_xStateExecuteExistent	BOOL	TRUE: **Execute** state exists
vq_xStateStoppingExistent	BOOL	TRUE: **Stopping** state exists
vq_xStateAbortingExistent	BOOL	TRUE: **Aborting** state exists
vq_xStateAbortedExistent	BOOL	TRUE: **Aborted** state exists
vq_xStateHoldingExistent	BOOL	TRUE: **Holding** state exists

Variable	Data type	Description
vq_xStateHeldExistent	BOOL	TRUE: **Held** state exists
vq_xStateUnholdingExistent	BOOL	TRUE: **Un-Holding** state exists
vq_xStateSuspendingExistent	BOOL	TRUE: **Suspending** state exists
vq_xStateUnsuspendingExistent	BOOL	TRUE: **Un-Suspending** state exists
vq_xStateResettingExistent	BOOL	TRUE: **Resetting** state exists
vq_xStateCompletingExistent	BOOL	TRUE: **Completing** state exists
vq_xStateCompleteExistent	BOOL	TRUE: **Complete** state exists
viq_diIndexAlarm	DINT	Index for active alarm table
viq_diIndexAlarmHistory	DINT	Index for active alarm history table
viq_diIndexProdConsumed	DINT	Index of consumed product
viq_diIndexProdProcessed	DINT	Index of processed product
viq_diIndexProdDefective	DINT	Index of not acceptable product
viq_diIndexAdminParameter	DINT	Index of admin parameter
viq_diIndexMode	DINT	Index of operation mode
viq_diIndexWarning	DINT	Index for active advisory table
vq_astWarning	ARRAY [0..9] OF ST_Alarm	Array of given size for active machine advisory code and messaging
vq_audiBgColorAlarm	ARRAY [0..10] OF UDINT	Variables to control the background color of the single alarms in display FR_Alarm *(see page 107)*.
vq_audiBgColorAlarmHistory	ARRAY [0..10] OF UDINT	Variables to control the background color of the single alarms in display FR_AlarmHistory *(see page 108)*.
vq_audiBgColorAlarmWarning	ARRAY [0..9] OF UDINT	Variables to control the background color of the single advisory in display FR_Warning *(see page 117)*.

Used By

- FB_VisController
- Visualization Frames

Section 3.3
SubStructures

What Is in This Section?

This section contains the following topics:

Topic	Page
ST_Alarm	54
ST_CountDescrip	55
ST_Descriptor	56
ST_EquipmentInterlock	57
ST_Ingredient	58
ST_Interface	59
ST_Product	60
ST_StateInfo	61
ST_Timestamp	62

ST_Alarm

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_Alarm is a collection of tags needed to describe an event (for example, alarms or advisories).

Structure Elements

Variable	Data type	Description
ID	DINT	Unique identifier of the event
Value	DINT	Value associated to the event for adding details or to break down the event ID to greater detail.
Message	STRING[GPL.Gc_uiMaxLengthOfAlarmMessage]	Message associated to the event
TimeEvent(1)	ST_TimeStamp *(see page 62)*	Time stamp (ISO data type) the event occurred
TimeAck(2)	ST_TimeStamp	Time stamp the event was acknowledged
Trigger	BOOL	TRUE: Indicates that the event is active.
Category	DINT	Category of the event
DateTime	DateTimeArray *(see page 63)*	Date and time when the event has occurred
AckDateTime	DateTimeArray	Date and time when the event was acknowledged

(1) Variable has been replaced by DateTime in ANSI/ISA TR88.00.02-2015 but remains in this library for compatibility reasons.

(2) Variable has been replaced by AckDateTime in ANSI/ISA TR88.00.02-2015 but remains in this library for compatibility reasons.

Used By

- FC_SetAlarm
- ST_Administration
- ST_VisInterface

ST_CountDescrip

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_CountDescrip is a collection of tags used to describe a counter value.

Structure Elements

Variable	Data type	Description
ID	DINT	ID of CountDescrip
Name	STRING[GPL.Gc_uiMaxLengthOfParameterName]	Name of CountDescrip
Unit	STRING[5]	String of CountDescrip units
Count	DINT	Count value of CountDescrip
AccCount	DINT	Accumulative count value of CountDescrip

Used By

- ST_Administration

ST_Descriptor

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure `ST_Descriptor` is the data type used for the parameter tags of the PackTags. It serves as descriptor of the respective parameter tag.

Structure Elements

Variable	Data type	Description
`ID`	DINT	ID of parameter
`Name`	STRING[`GPL.Gc_uiMaxLengthOfParameterName`]	Name of parameter
`Unit`	STRING[5]	String of parameter units
`Value`	REAL	Value of parameter

Used By

- `ST_Administration`
- `ST_Commmand`
- `ST_Status`
- `ST_Ingredient`
- `ST_Interface`
- `ST_Product`

ST_EquipmentInterlock

Overview

Type	Data structure
Available as of	V1.1.0.0

Description

The structure `ST_EquipmentInterlock` indicates whether the downstream and upstream system is the reason for the suspended state of a machine.

Structure Elements

Variable	Data type	Description
`Blocked`	BOOL	TRUE: Indicates that a downstream system is not able to accept product.
`Starved`	BOOL	TRUE: Indicates that an upstream system is not able to supply product

Used By

- `ST_Status`

ST_Ingredient

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure `ST_Ingredient` is a collection of tags used to describe the raw materials that are needed for the product.

Structure Elements

Variable	Data type	Description
`IngredientID`	DINT	Arbitrary number associated with the raw material, or ingredient for a particular product number.
`Parameter`	ARRAY [0..`Gc_uiNumberOfIngredientParameter` *(see page 69)*] OF `ST_Descriptor` *(see page 56)*	Structured array describes the specific ingredients or raw materials, which are needed or were used by the machine to process the specified product.

Used By

- `ST_Product`

ST_Interface

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_Interface is a collection of tags that are used to describe communication command values between machines using the PackTag structures.

Structure Elements

Variable	Data type	Description
Number	DINT	Identification number of upstream or downstream machine
ControlCmdNumber	DINT	Control command number for upstream or downstream machine
CmdValue	DINT	Command value associated with the control command number.
Parameter	ARRAY [0..GPL.Gc_uiNumOfParam *(see page 69)*] OF ST_Descriptor	Array of ST_Descriptor *(see page 56)*, parameter tags associated to the remote interface

Used By

- ST_Commmand
- ST_Status

ST_Product

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_Product is a collection of tags used to describe the product that the machine is making.

Structure Elements

Variable	Data type	Description
ProductID	DINT	ID of the product, used to indicate to the machine which product it is producing.
ProcessVariable	ARRAY [0..GPL.Gc_uiNumberOfProcessVariable] OF ST_Descriptor	Structured array describes the specific process variables, which are needed or were used by the machine to process the specified product.
Ingredients	ARRAY [0..GPL.Gc_uiNumberOfIngredients *(see page 69)*] OF ST_Ingredient *(see page 58)*	Structured array, holds the information about the raw materials, which are needed or were used by the machine to process the specified product.

Used By

- ST_Commmand
- ST_Status

ST_StateInfo

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_StateInfo indicates whether the state is defined for the given operation mode and whether a change of operation mode is possible. The information is used to initialize the operation mode manager.

Structure Elements

Variable	Data type	Description
xStateExistent	BOOL	TRUE: Indicates that the state is defined for the given operation mode.
xModeChangePossible	BOOL	TRUE: indicates that a change of operation mode is possible while in this state.

Used By

- ST_UnitModeDefinition

ST_Timestamp

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure ST_TimeStamp contains the date and time.

NOTE: The structure has become obsolete with the 2015 revision of ANSI/ISA TR88.00.02. However, it is kept in the library in order to stay compatible with applications which have been created with the earlier versions of this library.

Structure Elements

Variable	Data type	Description
AlmDate	DATE	Contains the timespan in seconds since 01.01.1970, 00:00 clock.
AlmTime	TOD	Contains the timespan in milliseconds since 00:00 clock.

Used By

- ST_Administration
- ST_Alarm

Chapter 4
Aliases - DateTimeArray

DateTimeArray

Overview

Type	Data structure
Available as of	V1.1.0.0

Description

The alias DateTimeArray represents a complex data type which is used in this library.

Structure Elements

Variable	Data type	Data type
DateTimeArray	ARRAY [0..6] OF DINT	The data elements associated with date and time shall be in the format per ISO 8601. [0] = year, [1] = month, [2] = day, [3] = hour, [4] = min, [5] = sec, [6] = µs (fraction of seconds)

Part III
Global Variables

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
5	Global Constants List	67
6	Global Parameter List	69

Chapter 5
Global Constants List

GCL

Overview

Type:	Global constants
Available as of:	V1.0.1.0

Description

The global constants list contains the global constants of the PackML library.

Global Constants

Variable	Data type	Value	Description
`Gc_sLibraryVersion`	STRING[80]	Vx.x.x.0[1]	Library version
`Gc_uiMaxNumberOfStates`	UINT	17	Maximum number of states
[1] This value varies to indicates the version of the library.			

Chapter 6
Global Parameter List

GPL

Overview

Type:	Global parameters
Available as of:	V1.0.1.0

Description

The global parameter list contains the global parameters of the PackML library which can be overwritten project-specifically in the library manager.

Global Parameters

Variable	Data type	Value	Description
`Gc_uiNumberOfInterface`	UINT[1..65535]	2	Number of remote interfaces, determines the array `ST_Command.RemoteInterface`
`Gc_uiNumOfParam`	UINT[1..65535]	10	Number of parameters for an interface, determines the array `ST_Interface.Parameter`
`Gc_uiNumOfAdminParameter`	UINT[1..65535]	100	Number of parameters for the local interface, determines the array `ST_Administration.Parameter` .
`Gc_uiNumberOfParameter`	UINT[1..65535]	50	Number of parameters in status and command structure, determines the arrays `ST_Status.Parameter` and `ST_Command.Parameter`
`Gc_uiNumberOfProducts`	UINT[1..65535]	3	Number of products which can be processed by the machine, determines the arrays `ST_Status.Product` and `ST_Command.Product`

Variable	Data type	Value	Description
Gc_uiNumberOfConsumedProducts	UINT[1..65535]	10	Number of products consumed by the machine, determines the array ST_Administration.ProdConsumedCount
Gc_uiNumberOfProcessedProducts	UINT[1..65535]	10	Number of products processed by the machine, determines the array ST_Administration.ProdProcessedCount
Gc_uiNumberOfDefectiveProducts	UINT[1..65535]	10	Number of products that are marked as not acceptable in the production machine, determines the array ST_Administration.ProdDefectiveCount
Gc_uiNumberOfProcessVariable	UINT[1..65535]	50	Number of process variables per product, determines the array ST_Product.ProcessVariable
Gc_uiNumberOfIngredientParameter	UINT[1..65535]	50	Number of parameters per ingredient, determines the array ST_Ingredients.Parameter
Gc_uiNumberOfIngredients	UINT[1..65535]	10	Number of ingredients per product, determines the array ST_Product.Ingredients
Gc_uiMaxNumberOfAlarms	UINT[1..65535]	100	Maximum number of alarms, determines the array ST_Administration.Alarm
Gc_uiNumberOfAlarmHistory	UINT[1..65535]	100	Maximum number of alarms in history, determines the array ST_Administration.AlarmHistory
Gc_uiMaxNumberOfModes	UINT[1..65535]	10	Maximum number of modes
Gc_uiMaxLengthOfParameterName	UINT[1..80]	80	Maximum length of parameter name in ST_Descriptor
Gc_uiMaxNumberOfWarnings	UINT[1..65535]	100	Maximum number of advisories, determines the array ST_Administration.Warning
Gc_uiMaxLengthOfAlarmMessage	UINT[1..255]	80	Maximum length of an alarm message specified with ST_Alarm
Gc_udiBgColorAlarmActive	UDINT	16#FFCFCF	Used for visualization, RGB color code for active alarms, 16#FFCFCF = light red

Variable	Data type	Value	Description
Gc_udiBgColorAlarmInactive	UDINT	16#FFFF80	Used for visualization, RGB color code for inactive and not acknowledged alarms, 16#FFFF80 = light yellow
Gc_udiBgColorAlarmAcknowledged	UDINT	16#AAD7FF	Used for visualization, RGB color code for inactive and acknowledged alarms, 16#AAD7FF = light blue
Gc_udiBgColorAlarmHistoryUnacknowledged	UDINT	16#FFFF80	Used for visualization, RGB color code for unacknowledged alarms in history, 16#FFFF80 = light yellow
Gc_udiBgColorWarningActive	UDINT	16#FFCFCF	Used for visualization, RGB color code for active advisories, 16#FFCFCF = light red
Gc_udiBgColorWarningInactive	UDINT	16#FFFF80	Used for visualization, RGB color code for inactive and not acknowledged advisories, 16#FFFF80 = light yellow
Gc_udiBgColorWarningAcknowledged	UDINT	16#AAD7FF	Used for visualization, RGB color code for inactive and acknowledged advisories, 16#AAD7FF = light blue

Part IV
Program Organization Units (POU)

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
7	Function Blocks	75
8	Functions	85

Chapter 7
Function Blocks

What Is in This Chapter?

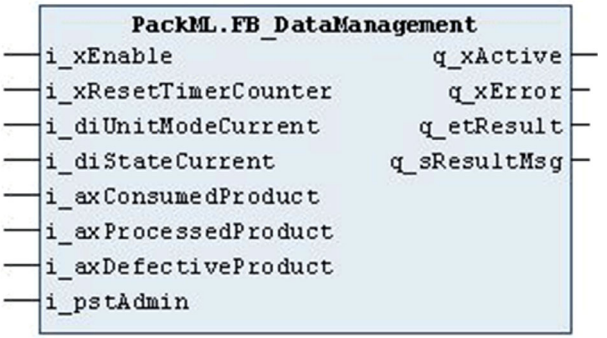
This chapter contains the following topics:

Topic	Page
FB_DataManagement	76
FB_ModeManager	79
FB_VisController	82

FB_DataManagement

Overview

Type:	Function block
Available as of:	V1.0.1.0



Functional Description

The function block FB_DataManagement is used to record certain machine data and to update the associated tags within the ST_Administration *(see page 39)*. The global structure of type ST_Administration is passed to the function block through the input i_pstAdmin.

Depending on the present unit mode (i_diUnitModeCurrent) and the machine state (i_diStateCurrent), the corresponding timer values are continuously updated. The values for control mode and machine state correspond to the definitions in the enumerations ET_Modes *(see page 31)* and ET_States *(see page 34)*.

The different product counters are triggered by the corresponding signals within the arrays which are provided by the inputs i_ax…Product.

A global reset of the timer and counter values can be triggered through the input i_xResetTimerCounter.

NOTE: The counter values provided by the tag AccCount within the structure ST_CountDescrip are given as non-resetting counters, hence they are not reset by this function block.

The retaining of data across external events, for example, a warm start of the controller, is not covered by this function block.

The following administration tags are processed by the function block:

- ST_Administration.ModeCurrentTime[#]
- ST_Administration.ModeCummulativeTime[#]

- ST_Administration.StateCurrentTime[#,#]
- ST_Administration.StateCummulativeTime[#,#]
- ST_Administration.ProdConsumedCount[#]
- ST_Administration.ProdProcessedCount[#]
- ST_Administration.ProdDefectiveCount[#]
- ST_Administration.AccTimeSinceReset
- ST_Administration.AlarmExtent
- ST_Administration.AlarmHistoryExtent
- ST_Administration.WarningExtent
- ST_Administration.PLCDateTime
- ST_Administration.PACDateTime

Interface

Input	Data type	Description
i_xEnable	BOOL	Activation and initialization of the function block.
i_xResetTimerCounter	BOOL	Upon a rising edge, the timer and counter values are reset.
i_diUnitModeCurrent	DINT	The present operation mode is passed to the function block via this input. Status.UnitModeCurrent should be applied to the input.
i_diStateCurrent	DINT	The present machine state is passed to the function block via this input. Status.StateCurrent should be applied to the input.
i_axConsumedProduct	ARRAY [1..GPL.Gc_uiNumberOfConsumedProducts] OF BOOL	Upon a rising edge of a boolean variable in the array, the corresponding counter value in the administration tag is increased.
i_axProcessedProduct	ARRAY [1..GPL.Gc_uiNumberOfConsumedProducts] OF BOOL	Upon a rising edge of a boolean variable in the array, the corresponding counter value in the administration tag is increased.
i_axDefectiveProduct	ARRAY [1..GPL.Gc_uiNumberOfDefectiveProducts] OF BOOL	Upon a rising edge of a boolean variable in the array, the corresponding counter value in the administration tag is increased.
i_pstAdmin	POINTER TO ST_Administration	Through this input, the pointer address to the administration tags is passed to the function block.

To prevent access violation eventually caused by pointer access to the memory, make sure that the pointer points to a variable of type ST_Administration.

Executing the command **Online Change** can move variables to another place in the memory. The shift of variables may have the effect that `POINTER` variables point to invalid memory. So, ensure that a pointer is not kept between cycles, but is reassigned in each cycle.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Assign the value of any `POINTER TO` type variable(s) prior to the first use of it within a function block, and at every subsequent cycle.

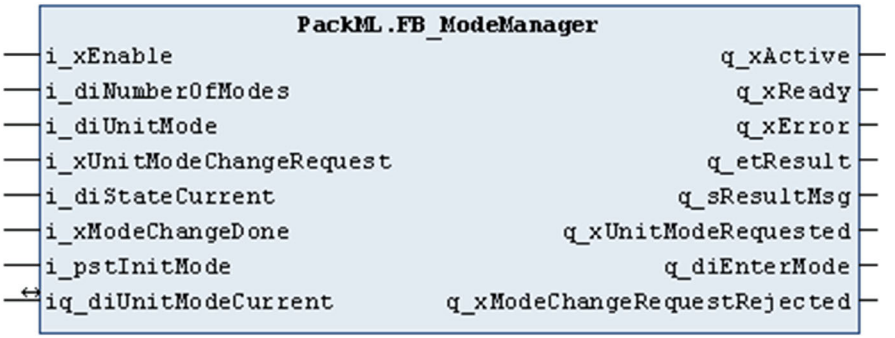
Failure to follow these instructions can result in death, serious injury, or equipment damage.

Output	Data type	Description
`q_xActive`	BOOL	If this output is set to TRUE, the function block is active.
`q_xError`	BOOL	If this output is set to TRUE, an error was detected. Refer to `ET_Result` *(see page 32)*.
`q_etResult`	`ET_Result`	Enumeration with the result.
`q_sResultMsg`	STRING[80]	Additional result message.

FB_ModeManager

Overview

Type:	Function block
Available as of:	V1.0.1.0



Functional Description

The function block FB_ModeManager can be used to manage mode change requests in a machine application, which is based on the PackML state model *(see page 24)*. Based on the unit mode definition the function block verifies whether the requested mode change is possible or not.

The unit mode definition comprises the definition of the possible states for each available control mode and the definition, from which state a mode change is permitted. For executing the unit mode definition, the auxiliary functions FC_InitStateModelStateChange *(see page 90)* and FC_InitStateModelExistingStates *(see page 93)* can be used.

Finally, the unit mode definition must be provided in an array of type ST_UnitModeDefinition. The indexes of the array correspond to the numeric value of the available control modes (refer to ET_Modes *(see page 31)*).

The unit mode definition must be made at start (initialization phase) of an application, prior to the first call with i_xEnable = TRUE of the instance of the function block FB_ModeManager.

Upon a rising edge of the input i_xEnable, the initialization phase of the function block FB_ModeManager is executed. During this phase, the unit state definitions for each available control mode are verified whether a PackML compliant state model can be created. If the initialization is completed successfully, the function block is ready for operation, which is indicated by the output q_xReady.

During initialization, the following conditions are verified:

- The maximum number of states is 17.
- The states **Execute** and **Stopped** are obligatory states for each control mode.
- The state **Completing** requires the state **Complete** too.
- The states **Holding** and **Unholding** require in each case the state **Held** too.
- The states **Suspending** and **UnSuspending** require in each case the state **Suspended** too.
- The states **Aborting** and **Clearing** require in each case the state **Aborted** too.
- The states **Resetting** and **Idle** can be used only in combination.

During normal operation, the function block verifies the mode change request, which is indicated by the inputs `i_diUnitMode` and `i_xUnitModeChange`, in relation to the present control mode (`iq_diUnitModeCurrent`) and the present state (`i_diStateCurrent`). Exclusively a rising edge on `i_sUnitModeChange` triggers an execution of the function block. The result is indicated by the outputs `q_diEnterMode`, and `q_xModeChangeRequestRejected`. If the mode change is permitted, the output `q_xModeChangeRequestRejected` indicates FALSE and the output `q_diEnterMode` is updated with the value of the requested mode. If so, the required steps to change the control mode must be executed depending on your application. Afterwards the information that the mode change was executed must be passed to the function block through the input `i_xModeChangeDone`. If this input indicates TRUE, the variable linked to the input / output `iq_diUnitModeCurrent` is updated with the previously requested control mode.

Interface

Input	Data type	Description
`i_xEnable`	BOOL	Activation and initialization of the function block.
`i_diNumberOfModes`	DINT	Number of operation modes If the value is changed, a reinitialization of the function block is required.
`i_diUnitMode`	DINT	Requested operation mode The PackTag `Command.UnitMode` should be applied to the input.
`i_xUnitModeChangeRequest`	BOOL	Upon a rising edge, the function block verifies whether a change of operation mode is possible. The PackTag `Command.UnitModeChangeRequest` should be applied to the input.
`i_diStateCurrent`	DINT	Machine present state The PackTag `Status.StateCurrent` should be applied to the input.

Input	Data type	Description
i_xModeChangeDone	BOOL	Feedback from the application that the requested mode change has been executed. Upon a rising edge, the variable linked to iq_diUnitModeCurrent is updated accordingly.
i_pstInitMode	POINTER TO ST_UnitModeDefinition *(see page 46)*	Through this input, the pointer address to the unit mode definitions is passed to the function block.** The unit mode definition provides the state definitions for each available operation mode and must be provided in an array of ST_UnitModeDefinition. The indexes of the array correspond to the numeric value of the available control modes (refer to ET_Modes *(see page 31)*). Therefore the pointer must point to the index which is associated to the operation mode Producing, which is the first mode.

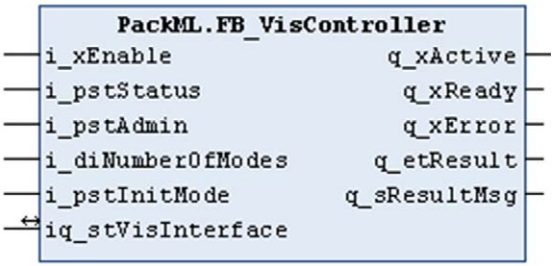
Input / output	Data type	Description
iq_diUnitModeCurrent	DINT	Through the variable linked to this input / output, the function block obtains the present operation mode on a change request and it writes the new operation mode after the change has been performed. The PackTag Status.UnitModeCurrent should be applied to the input/output.

Output	Data type	Description
q_xActive	BOOL	If this output is set to TRUE, the function block is active.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. Refer to ET_Result *(see page 32)*.
q_xReady	BOOL	If this output is set to TRUE, the function block is ready for operation.
q_etResult	ET_Result	Enumeration with the result.
q_sResultMsg	STRING[80]	Additional result message.
q_xUnitModeRequested	BOOL	This reflects the input i_xUnitModeChangeRequest. The PackTag Status.UnitModeRequested should be applied to this output.
q_diEnterMode	DINT	This indicates the operation mode which is to be changed to. If the function block is deactivated, the output is set to 0.
q_xModeChangeRequestRejected	BOOL	As long as this output indicates TRUE, a mode change request is not authorized by this function block and should not be performed.

FB_VisController

Overview

Type:	Function block
Available as of:	V1.0.1.0



Functional Description

The FB_VisController is used to transfer the data (PackTags) from the application into the data structure of type ST_VisInterface which is used by the visualization frames provided with the PackML library. Further, the function block implements functions to control the appearance of the visualization. For example: the scrolling through the alarm list and the control of state-dependent background colors for alarms.

Interface

Input	Data type	Description
i_xEnable	BOOL	Activation and initialization of the function block.
i_pstStatus	POINTER TO ST_Status	Through this input, the pointer address to the PackTags of type ST_Status *(see page 42)* is passed to the function block.
i_pstAdmin	POINTER TO ST_Administration	Through this input, the pointer address to the PackTags of type ST_Administration *(see page 39)* is passed to the function block.
i_diNumberOfModes	DINT	Number of operation modes. If the value is changed, a reinitialization of the function block is required.

Input	Data type	Description
i_pstInitMode	POINTER TO ST_UnitModeDefinition	Through this input, the pointer address to the unit mode definitions is passed to the function block.* * The unit mode definition provides the state definitions for each available operation mode and must be provided in an array of ST_UnitModeDefinition. The indexes of the array correspond to the numeric value of the available control modes (refer to ET_Modes *(see page 31)*). Therefore, the pointer must point to the index which is associated to the operation mode Producing, which is the first mode.

Input / Output	Data type	Description
iq_stVisInterface	ST_VisInterface *(see page 48)*	Through this input / output, the variable is passed to the function block which is linked to the visualizations provided in the PackML library.

Output	Data type	Description
q_xActive	BOOL	If this output is set to TRUE, the function block is active.
q_xReady	BOOL	If this output is set to TRUE, the function block is ready for operation.
q_xError	BOOL	If this output is set to TRUE, an error was detected.
q_etResult	ET_Result	Result. Refer to ET_Result *(see page 32)*.
q_sResultMsg	STRING[80]	Additional result message.

Chapter 8
Functions

What Is in This Chapter?

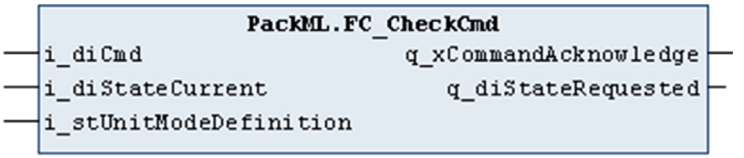
This chapter contains the following topics:

Topic	Page
FC_CheckCmd	86
FC_EtResultToString	88
FC_GetDateTimeAsArray	89
FC_InitStateModelChangeStates	90
FC_InitStateModelExistingStates	93
FC_SetAlarm	95
FC_SetWarning	97

FC_CheckCmd

Overview

Type:	Function
Available as of:	V1.0.4.0



Functional Description

The function `FC_CheckCmd` is used to verify the plausibility of a state transition command (`i_diCmd`) in the present state (`i_diStateCurrent`) of the machine. The plausibility verification is performed based on the unit mode definition (`i_stUnitModeDefinition`), which is provided in an array of type `ST_UnitModeDefinition`. It is assumed that the unit mode definition has been validated (and deemed correct) by the function block `FB_ModeManager`.

If the command is deemed as plausible, the output `q_xCommandAcknowledge` indicates TRUE and the target state is provided on the output `q_diStateRequested`.

The following table indicates which commands are supported in which state and what is the resulting target state, which was derived from the respective unit state definition.

Initial state	Transition command	Target state
-	`Undefined`	Undefined, the output `q_xCommandAcknowledge` is FALSE.
Execute	`Reset`	**Resetting** if **Complete** and **Completing** do not exist.
Complete	`Reset`	**Resetting**, or **Starting**, if **Resetting** does not exist, or **Execute**, if **Resetting** and **Starting** does not exist.
Stopped		
Idle	`Start`	**Starting**, or **Execute**, if **Starting** does not exist.
Stopped	`Start` (command is only processed if **Idle** does not exist)	**Starting**, or **Execute**, if **Starting** does not exist.
Any state except **Aborted**, **Aborting**, **Clearing**, **Stopping**, and **Stopped**	`Stop`	**Stopping**, or **Stopped**, if **Stopping** does not exist.

Initial state	Transition command	Target state
Execute	`Hold`	**Holding**, or **Held** if **Holding** does not exist.
Held	`UnHold`	**Un-Holding**, or **Execute** if **Un-Holding** does not exist.
Execute	`Suspend`	**Suspending**, or **Suspended** if **Suspending** does not exist.
Suspended	`UnSuspend`	**Un-Suspending**, or **Execute** if **Un-Suspending** does not exist.
Any state except **Aborted** and **Aborting**	`Abort`	**Aborting**, or **Aborted** if **Aborting** does not exist.
Aborted	`Clear`	**Clearing**, or **Stopped** if **Clearing** does not exist.

Interface

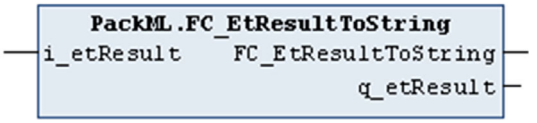
Inputs	Data type	Description
`i_diCmd`	DINT	The state transition command; the PackTag `ST_Command.CntrlCmd` should be applied to the input.
`i_diStateCurrent`	DINT	Machine present state; the PackTag `ST_Status.StateCurrent` should be applied to the input.
`i_stUnitModeDefinition`	`ST_UnitModeDefinition`	The operation mode definition of the present operation mode; the PackTag `ST_Status.UnitModeCurrent` should be used as index variable of array of `ST_UnitModeDefinition`.

Output	Data type	Description
`q_xCommandAcknowledge`	BOOL	Indicates TRUE if the given command is plausible.
`q_diStateRequested`	DINT	Provides the target state if the given command is plausible.

FC_EtResultToString

Overview

Type:	Function
Available as of:	V1.0.1.0



Task

Convert an enumeration element of type `ET_Result` to a string value.

Functional Description

Using the function `FC_EtResultToString`, you can convert an enumeration element of type `ET_Result` to a string value.

Interface

Input	Data type	Description
`i_etResult`	`ET_Result`	Enumeration with the result.

Output	Data type	Description
`q_etResult`	`ET_Result`	Enumeration with the result.

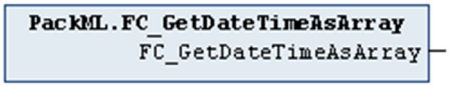
Return Value

Data type	Description
STRING(80)	The `ET_Result` converted to a string value.

FC_GetDateTimeAsArray

Overview

Type:	Function
Available as of:	V1.1.0.0



Functional Description

The function `FC_GetDateTimeAsArray` is used to obtain the RTC of the controller and to convert it to the data type `DateTimeArray` *(see page 63)*.

The return value of the function is of type `DateTimeArray` and represents the value of the RTC of the controller.

FC_InitStateModelChangeStates

Overview

Type:	Function
Available as of:	V1.0.1.0



Task

Use the function `FC_InitStateModelChangeStates` to define, for an operation mode, from which state a change of operation mode is possible.

Functional Description

Using the input/output `iq_stStateModel`, a structure of type `ST_UnitModeDefinition` is passed to the function.

Use the 17 inputs of type BOOL, which reflect the states of an operation mode, to define from which states a change of operation mode is possible.

TRUE indicates that a change of operation mode is possible; FALSE indicates that a change of operation mode must not be attempted from this state.

Following successful initialization of the operation mode, the function provides a TRUE.

Interface

Input	Data type	Description
i_xStopping	BOOL	If this input is set to TRUE, the **Stopping** state is allowed to change the operation mode.
i_xStopped	BOOL	If this input is set to TRUE, the **Stopped** state is allowed to change the operation mode.
i_xResetting	BOOL	If this input is set to TRUE, the **Resetting** state is allowed to change the operation mode.
i_xIdle	BOOL	If this input is set to TRUE, the **Idle** state is allowed to change the operation mode.
i_xStarting	BOOL	If this input is set to TRUE, the **Starting** state is allowed to change the operation mode.
i_xExecute	BOOL	If this input is set to TRUE, the **Execute** state is allowed to change the operation mode.
i_xHolding	BOOL	If this input is set to TRUE, the **Holding** state is allowed to change the operation mode.
i_xHeld	BOOL	If this input is set to TRUE, the **Held** state is allowed to change the operation mode.
i_xUnHolding	BOOL	If this input is set to TRUE, the **Un-Holding** state is allowed to change the operation mode.
i_xSuspending	BOOL	If this input is set to TRUE, the **Suspending** state is allowed to change the operation mode.
i_xSuspended	BOOL	If this input is set to TRUE, the **Suspended** state is allowed to change the operation mode.
i_xUnSuspending	BOOL	If this input is set to TRUE, the **Un-Suspending** state is allowed to change the operation mode.
i_xCompleting	BOOL	If this input is set to TRUE, the **Completing** state is allowed to change the operation mode.
i_xComplete	BOOL	If this input is set to TRUE, the **Complete** state is allowed to change the operation mode.
i_xAborting	BOOL	If this input is set to TRUE, the **Aborting** state is allowed to change the operation mode.
i_xAborted	BOOL	If this input is set to TRUE, the **Aborted** state is allowed to change the operation mode.
i_xClearing	BOOL	If this input is set to TRUE, the **Clearing** state is allowed to change the operation mode.

Input / output	Data type	Description
`iq_stStateModel`	`ST_UnitModeDefinition`	This structure reflects the operation mode which is to be initialized.

Return Value

Data type	Description
BOOL	TRUE if initialization of the operation mode was successful.

FC_InitStateModelExistingStates

Overview

Type:	Function
Available as of:	V1.0.1.0



Task

Use the `FC_InitStateModelExistingStates` function to initialize the state model and define an operation mode. By presenting TRUE or FALSE at the inputs, you are selecting those states to be part of your state model.

Functional Description

Using the input/output `iq_stStateModel`, a structure of type `ST_UnitModeDefinition` is passed to the function.

This structure reflects an operation mode. Using the 18 inputs of type BOOL which reflect the existence of an operation mode and its states, it is possible to define which states make up the relevant operation mode.

TRUE signifies that the state exists; FALSE signifies that the state does not exist.

Following successful initialization of the operation mode, the function provides a TRUE.

Interface

Input	Data type	Description
i_xModeExistent	BOOL	If this input is set to TRUE, the operation mode is available.
i_xStopping	BOOL	If this input is set to TRUE, the **Stopping** state is available.
i_xStopped	BOOL	If this input is set to TRUE, the **Stopped** state is available.
i_xResetting	BOOL	If this input is set to TRUE, the **Resetting** state is available.
i_xIdle	BOOL	If this input is set to TRUE, the **Idle** state is available.
i_xStarting	BOOL	If this input is set to TRUE, the **Starting** state is available.
i_xExecute	BOOL	If this input is set to TRUE, the **Execute** state is available.
i_xHolding	BOOL	If this input is set to TRUE, the **Holding** state is available.
i_xHeld	BOOL	If this input is set to TRUE, the **Held** state is available.
i_xUnHolding	BOOL	If this input is set to TRUE, the **Un-Holding** state is available.
i_xSuspending	BOOL	If this input is set to TRUE, the **Suspending** state is available.
i_xSuspended	BOOL	If this input is set to TRUE, the **Suspended** state is available.
i_xUnSuspending	BOOL	If this input is set to TRUE, the **Un-Suspending** state is available.
i_xCompleting	BOOL	If this input is set to TRUE, the **Completing** state is available.
i_xComplete	BOOL	If this input is set to TRUE, the **Complete** state is available.
i_xAborting	BOOL	If this input is set to TRUE, the **Aborting** state is available.
i_xAborted	BOOL	If this input is set to TRUE, the **Aborted** state is available.
i_xClearing	BOOL	If this input is set to TRUE, the **Clearing** state is available.

Input / output	Data type	Description
iq_stStateModel	ST_UnitModeDefinition	This structure reflects the operation mode which is to be initialized.

Return Value

Data type	Description
BOOL	TRUE if initialization of the operation mode was successful.

FC_SetAlarm

Overview

Type:	Function
Available as of:	V1.0.1.0



Functional Description

The function `FC_SetAlarm` is an auxiliary function to write alarm messages into the appropriate administration tag and also delete them again from the same. An alarm is identified exclusively by its unique identifier and the associated value. The identifier and its value are specified by the input/output parameter `iq_stSetAlarm` which corresponds with the input/output parameter `iq_astAlarms[#]`.

This function cannot be used simultaneously in several tasks.

Through the input `iq_stSetAlarm`, a machine-specific alarm message can be passed to the function.

Through the input/output parameter `iq_astAlarms`, the administration tag `Admin.Alarm[#]` must be passed to the function. It represents the list of active alarms in the machine unit in chronological order starting with the first occurred and still active alarm.

Through the input/output parameter `iq_stAlarmHistory`, the administration tag `Admin.AlarmHistory[#]` must be passed to the function. It represents the alarm history of the machine unit containing the alarms in chronological order starting with the last reset alarm.

If the function is called with input `i_xSetAlarm = TRUE`, the function verifies whether the specified alarm message is already in the list of active alarms. If not, the function obtains the RTC of the controller and writes it together with the specified alarm message into the list of active alarms. If the maximum number of messages in the list is reached, no new message is added.

If the function is called with `i_xSetAlarm = FALSE`, the specified alarm message is removed from the list of active alarms. Then it is written at the first position of the alarm history while the existing entries are moved one position down in the list. If the list is full, the oldest reset alarm is removed.

If the function is called with inputs `i_xAckAlarm` and `i_xSetAlarm = TRUE`, the function obtains the RTC of the controller and updates the parameter (tag) `AckDateTime` for the specified alarm message in the list of active alarms. If an alarm has been acknowledged once, a new request to acknowledge has no effect.

The return value of the function indicates TRUE if the specified alarm has been either written into the list or deleted from the list, or has been set to acknowledged. If the function returns FALSE, either no action was requested or the maximum number of messages in the list is reached.

Interface

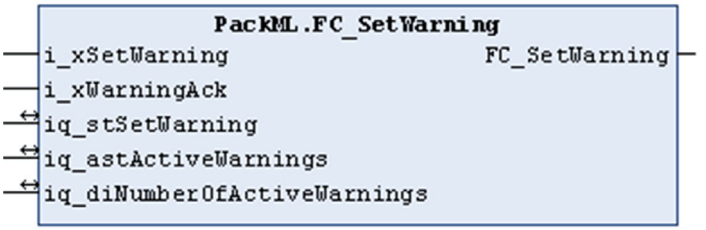
Input	Data type	Description
`i_xSetAlarm`	BOOL	• TRUE: the reason code or machine-specific alarm message passed to the input/output `iq_stSetAlarm` is written into the tag at the input/output `iq_stActiveAlarm`. • FALSE: the message is deleted from `iq_stActiveAlarms` and entered `iq_stAlarmHistory`.
`i_xAlarmAck`	BOOL	TRUE: the message has been detected. The appropriate time stamp for `Admin.Alarm[#].TimeAck` is determined.

Input/Output	Data type	Description
`iq_stSetAlarm`	`ST_InitAlarm` *(see page 45)*	Reason code or machine-specific alarm message is passed to this input/output.
`iq_astActiveAlarms`	ARRAY [1..`Gc_uiMaxNumberOfAlarms` *(see page 69)*] OF `ST_Alarm` *(see page 54)*	The administration tag `Admin.Alarm[#]` should be applied to this input/output.
`iq_astAlarmHistory`	ARRAY [1..`Gc_uiNumberOfAlarmHistory` *(see page 69)*] OF `ST_Alarm` *(see page 54)*	The administration tag `Admin.AlarmHistory[#]` should be applied to this input/output:
`iq_diNumberOfActiveAlarms`	DINT	Provides the number of active alarm messages.

FC_SetWarning

Overview

Type:	Function
Available as of:	V1.1.0.0



Functional Description

The function `FC_SetWarning` is an auxiliary function to write advisory messages into the appropriate administration tag and also delete them again from the same. An advisory is identified exclusively by its unique identifier and the associated value. The identifier and its value are specified by the input/output parameter `iq_stSetWarning` which corresponds with the input/output parameter `iq_astWarnings[#]`.

This function cannot be used simultaneously in several tasks.

Through the input `iq_stSetWarning`, a machine-specific advisory message can be passed to the function.

Through the input/output parameter `iq_astWarnings`, the administration tag `Admin.Warning[#]` must be passed to the function. It represents the list of active advisories in the machine unit in chronological order starting with the first occurred and still active advisory.

If the function is called with input `i_xSetWarning = TRUE`, the function verifies whether the specified advisory message is already in the list of active advisories. If not, the function obtains the RTC of the controller and writes it together with the specified advisory message into the list of active advisories. If the maximum number of messages in the list is reached, no new messages are added.

If the function is called with `i_xSetWarning = FALSE`, the specified advisory message is removed from the list of active advisories.

If the function is called with inputs `i_xAckWarning` and `i_xSetWarning = TRUE`, the function obtains the RTC of the controller and updates the parameter (tag) `AckDateTime` for the specified advisory message in the list of active advisories. If an advisory has been acknowledged once, a new request to acknowledge has no effect.

The return value of the function indicates TRUE if the specified advisory has been either written into the list or deleted from the list, or has been set to acknowledged. If the function returns FALSE, either no action was requested or the maximum number of messages in the list is reached.

Interface

Input	Data type	Description
`i_xSetWarning`	BOOL	Request to write (TRUE) or remove (FALSE) the specified message into/from the list linked to `iq_astWarnings`.
`i_xWarningAck`	BOOL	Request to acknowledge an active advisory.

Input/Output	Data type	Description
`iq_stSetWarning`	`ST_InitAlarm` *(see page 45)*	Specifies the advisory message to be treated by the function.
`iq_astWarnings`	ARRAY [1..`Gc_uiMaxNumberOfWarnings` *(see page 69)*] OF `ST_Alarm` *(see page 54)*	The administration tag `Admin.Warning[#]` must be linked to this input/output.
`iq_diNumberOfActiveWarnings`	DINT	Provides the number of advisories in the list.

Part V
Visualization

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
9	BackgroundFrames	101
10	States Frames	103
11	Visualization Frames	105

Chapter 9
BackgroundFrames

FR_<BackgroundFrame>

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display a frame in the background.

Functional Description

The FR_<BackgroundFrame> *(see page 101)* visualization frames (for example, FR_Control1024x768) are frames in the background to group the other frames, and to realize control and navigation functionalities. See examples below.

The following background frames are available:

Frame	Description
FR_Control1024x768	Control frame, size 1024x768 pixels
FR_Nav1024x768	Navigation frame, size 1024x768 pixels
FR_NavControl1024x768	Navigation and control frame, size 1024x768 pixels
FR_Control800x600	Control frame, size 800x600 pixels
FR_Nav800x600	Navigation frame, size 800x600 pixels
FR_NavControl800x600	Navigation and control frame, size 800x600 pixels

Interface

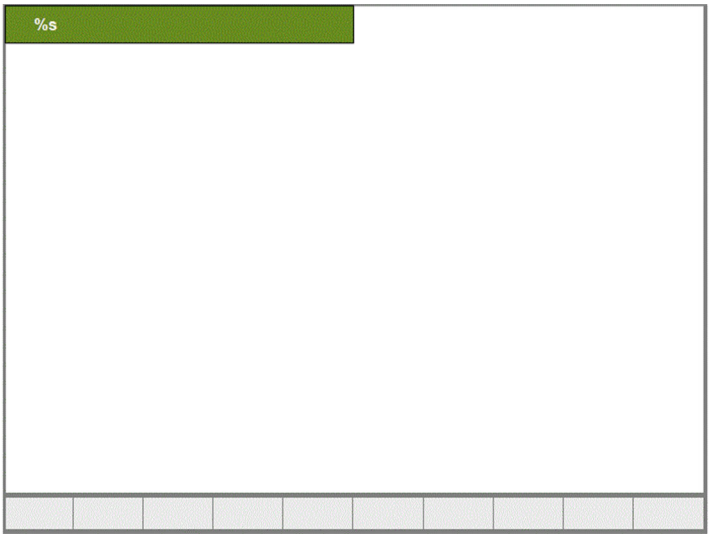
Input	Data type	Description
i_sTitle	STRING(80)	Title of the background frame

Examples

FR_Control1024x768 with controls on the left side



FR_Nav1024x768 with navigation buttons at the bottom



Chapter 10
States Frames

FR_<State>

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the single states of the state model for the operation mode.

Functional Description

For each specified state in the state model (refer to `ET_States` *(see page 34)*), a visualization is provided.

The frames are used by the `FR_DynStateModel` *(see page 110)* to display the single states of the state model. See following example.

Interface

Input / output	Data type	Description
`iq_stVisInterface`	`ST_VisInterface` *(see page 48)*	Interface to the `FB_VisController` *(see page 82)*

Example



Chapter 11
Visualization Frames

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
General	106
FR_Alarm	107
FR_AlarmHistory	108
FR_CurrentModeAndStateTime	109
FR_DynStateModel	110
FR_ModeAndStateTime	112
FR_ProdConsumedCount	114
FR_ProdDefectiveCount	115
FR_ProdProcessedCount	116
FR_Warning	117
FR_StopReason	118
FR_StatesDisabled	119

General

Overview

The library provides several visualization frames.

You can use these frames to display information about:

- Alarms
- Machine operation modes and states
- Production data

FB_VisController Function Block

The main task of the function block FB_VisController is to process data provided by the administration tags and status tags and to provide these to the visualization frames via the visualization interface input/output iq_stVisInterface of type ST_VisInterface.

Visualization Frames

- FR_Alarm *(see page 107)*
- FR_AlarmHistory *(see page 108)*
- FR_CurrentModeAndStateTime *(see page 109)*
- FR_DynStateModel *(see page 110)*
- FR_ModeAndStateTime *(see page 112)*
- FR_ProdConsumedCount *(see page 114)*
- FR_ProdDefectiveCount *(see page 115)*
- FR_ProdProcessedCount *(see page 116)*
- FR_Warning *(see page 117)*
- FR_StopReason *(see page 118)*
- FR_StatesDisabled *(see page 119)*

FR_Alarm

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display an alarm list of the active alarms.

Functional Description

FR_Alarm is a visualization frame to display the alarm list of the active alarms (Admin.Alarm[#]). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	ST_VisInterface *(see page 48)*	Interface to the FB_VisController *(see page 82)*

Example

#	ID	Value	Cat.	Message	Trigger	DateTime	AckDateTime
1	39	2	2	This alarm is active.	TRUE	2017 - 9 - 14 - 14 : 15 : 59 . 619000	2017 - 9 - 14 - 14 : 15 : 59 . 619000
2	1234	1	1	This alarm is inactive but not acknowledged.	FALSE	2017 - 9 - 14 - 14 : 36 : 23 . 2365	0 - 0 - 0 - 0 : 0 : 0 . 0
3	456	63	4	This alarm is inactive and acknowledged.	FALSE	2017 - 9 - 14 - 14 : 52 : 26 . 56987	2017 - 9 - 14 - 14 : 53 : 11 . 369
4	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 . 0	0 - 0 - 0 - 0 : 0 : 0 . 0
5	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 . 0	0 - 0 - 0 - 0 : 0 : 0 . 0
6	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 . 0	0 - 0 - 0 - 0 : 0 : 0 . 0
7	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 . 0	0 - 0 - 0 - 0 : 0 : 0 . 0
8	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 . 0	0 - 0 - 0 - 0 : 0 : 0 . 0
9	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 . 0	0 - 0 - 0 - 0 : 0 : 0 . 0
v	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 . 0	0 - 0 - 0 - 0 : 0 : 0 . 0

FR_AlarmHistory

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the alarm history.

Functional Description

FR_AlarmHistory is a visualization frame to display the alarm history (Admin.AlarmHistory[#]). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	ST_VisInterface *(see page 48)*	Interface to the FB_VisController *(see page 82)*

Example

#	ID	Value	Cat.	Message	DateTime	AckDateTime
1	38	1	19	This alarm is not yet acknowledged.	2017 - 9 - 14 - 14:15:57.549000	0 - 0 - 0 - 0:0:0.0
2	37	1	65	This alarm is already acknowledged.	2017 - 9 - 14 - 14:14:30.540000	2017 - 9 - 14 - 14:14:41.233
3	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
4	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
5	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
6	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
7	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
8	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
9	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
v	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0

FR_CurrentModeAndStateTime

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the time the machine has been running in the present operation mode/state.

Functional Description

FR_CurrentModeAndStateTime is a visualization frame to display the time the machine has been running in the present operation mode (Admin.ModeCurrentTime[#] & Admin.ModeCumulativeTime[#]) and state (Admin.StateCurrentTime[#,#] & Admin.StateCumulativeTime[#,#]). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	ST_VisInterface *(see page 48)*	Interface to the FB_VisController *(see page 82)*

Example

Current Unit Mode:	1
Time in Current Mode:	58
Cummulative Time in Current Mode:	4885
Current State:	6
Time in Current State:	40
Cummulative Time in Current State:	40

NOTE: This example only demonstrates the PackTag information, which, according to the standards, is presented as seconds.

FR_DynStateModel

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Dynamically display the visualization of the state model for the operation mode during online operation.

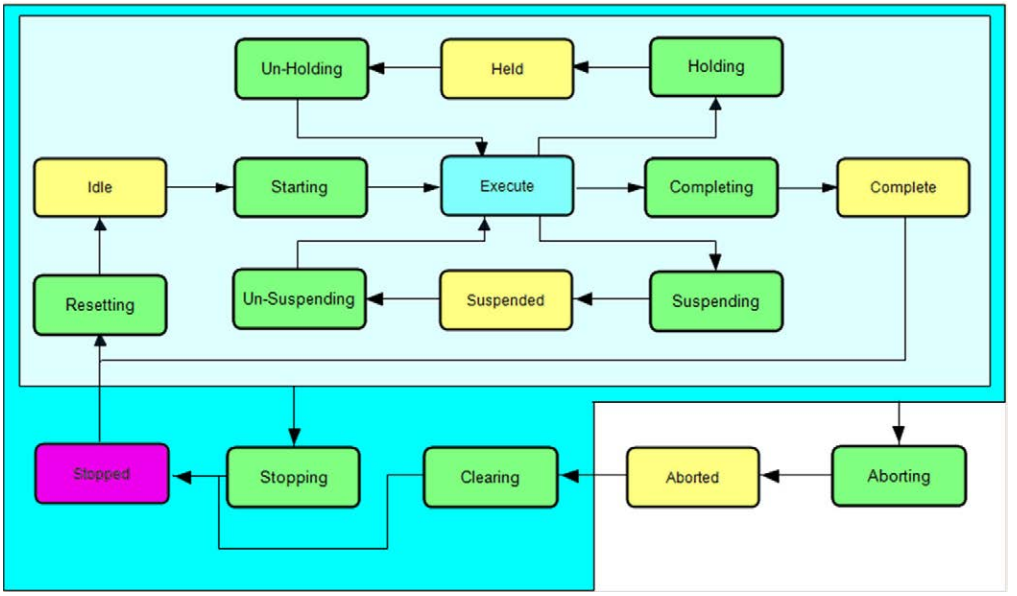
Functional Description

`FR_DynStateModel` provides a visualization frame to display the state models. This frame dynamically generates the visualization of the corresponding state model during online operation.

Interface

Input / output	Data type	Description
`iq_stVisInterface`	`ST_VisInterface` *(see page 48)*	Interface to the `FB_VisController` *(see page 82)*

Example


Un-Holding
Held
Holding
Idle
Starting
Execute
Completing
Complete
Resetting
Un-Suspending
Suspended
Suspending
Stopped
Stopping
Clearing
Aborted
Aborting

FR_ModeAndStateTime

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the operation time of all operation modes and their respective states.

Functional Description

FR_ModeAndStateTime is a visualization frame to display the operation time of all operation modes (Admin.ModeCurrentTime[#] & Admin.ModeCumulativeTime[#]) and their respective states (Admin.StateCurrentTime[#,#] & Admin.StateCumulativeTime[#,#]).

Interface

Input / output	Data type	Description
iq_stVisInterface	ST_VisInterface *(see page 48)*	Interface to the FB_VisController *(see page 82)*

Example

Unit Mode [0]			>
Current Time	0		
Cumulative Time	18		
States			
Current Time		Cumulative Time	
Undefined	0	Undefined	18
Execute	0	Execute	0
Stopping	0	Stopping	0
Stopped	0	Stopped	0
Resetting	0	Resetting	0
Idle	0	Idle	0
Starting	0	Starting	0
Holding	0	Holding	0
Held	0	Held	0
Unholding	0	Unholding	0
Suspending	0	Suspending	0
Suspended	0	Suspended	0
Unsuspending	0	Unsuspending	0
Completing	0	Completing	0
Complete	0	Complete	0
Aborting	0	Aborting	0
Aborted	0	Aborted	0
Clearing	0	Clearing	0

FR_ProdConsumedCount

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the product counters of the consumables.

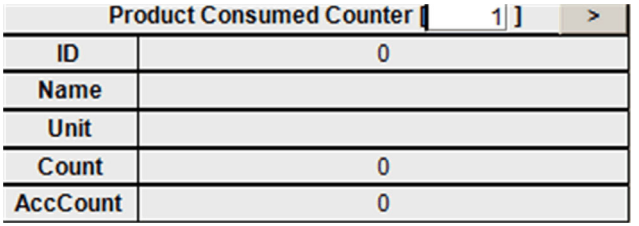
Functional Description

`FR_ProdConsumedCount` is a visualization frame to display the product counters of the consumables (`Admin.ProdConsumedCount[#]`).

Interface

Input / output	Data type	Description
`iq_stVisInterface`	`ST_VisInterface` *(see page 48)*	Interface to the `FB_VisController` *(see page 82)*

Example



FR_ProdDefectiveCount

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the product counters of the goods detected as not acceptable.

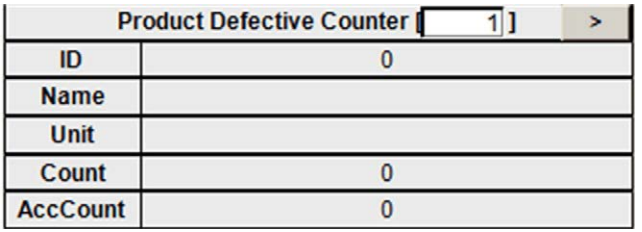
Functional Description

`FR_ProdDefectiveCount` is a visualization frame to display the product counters of the goods detected as not acceptable (`Admin.ProdDefectiveCount[#]`)

Interface

Input / output	Data type	Description
`iq_stVisInterface`	`ST_VisInterface` *(see page 48)*	Interface to the `FB_VisController` *(see page 82)*

Example



FR_ProdProcessedCount

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the product counters of the produced goods.

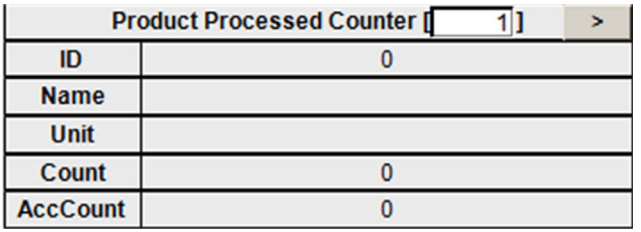
Functional Description

`FR_ProdProcessedCount` is a visualization frame to display the product counters of the produced goods (`Admin.ProdProcessedCount[#]`).

Interface

Input / output	Data type	Description
`iq_stVisInterface`	`ST_VisInterface` *(see page 48)*	Interface to the `FB_VisController` *(see page 82)*

Example



FR_Warning

Overview

Type:	Visualization frame
Available as of:	V1.1.0.0
Implements:	VisuElems.IVisualization

Task

Display a list of the active advisories.

Functional Description

FR_Warning is a visualization frame to display the list of the active advisories (Admin.Warning[#]). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	ST_VisInterface *(see page 48)*	Interface to the FB_VisController *(see page 82)*

Example

#	ID	Value	Cat.	Message	Trigger	DateTime	AckDateTime
1	123	1	1	This Warning is active.	TRUE	2017 - 9 - 14 - 14 : 58 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
2	456	2	2	This warning is inactive but not acknowledged.	FALSE	2017 - 14 - 9 - 14 : 58 : 1 .123	0 - 0 - 0 - 0 : 0 : 0 .0
3	789	3	3	This warning is inactive and acknowledged.	FALSE	2017 - 9 - 14 - 14 : 58 : 32 .1258	2017 - 9 - 14 - 14 : 59 : 1 .0
4	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
5	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
6	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
7	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
8	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
9	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
v	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0

FR_StopReason

Overview

Type:	Visualization frame
Available as of:	V1.1.0.0
Implements:	VisuElems.IVisualization

Task

Display the PackTag `Admin.StopReason`.

Functional Description

`FR_StopReason` is a visualization frame to display the PackTag `Admin.StopReason`.

Interface

Input / output	Data type	Description
`iq_stVisInterface`	`ST_VisInterface` *(see page 48)*	Interface to the `FB_VisController` *(see page 82)*

Example

ID	Value	Cat.	Message	DateTime
10	8	21	Abort command detetcted.	2017 - 10 - 23 - 13:15:26.532000

FR_StatesDisabled

Overview

Type:	Visualization frame
Available as of:	V1.1.0.0
Implements:	VisuElems.IVisualization

Task

Monitor and control the PackTag `Admin.StatesDisabled`.

Functional Description

`FR_StatesDisabled` is a visualization frame to monitor and control the PackTag `Admin.StatesDisabled`.

Interface

Input / output	Data type	Description
`iq_stVisInterface`	`ST_VisInterface` *(see page 48)*	Interface to the `FB_VisController` *(see page 82)*

Example



Glossary

R

RTC

(*real-time clock*) A battery-backed time-of-day and calender clock that operates continuously, even when the controller is not powered for the life of the battery.

Index

B

BackgroundFrames, *101*

D

DateTimeArray, *63*

E

ET_Cmd, *30*
- abort, *87*
- Abort, *30*
- clear, *87*
- Clear, *30*
- hold, *87*
- Hold, *30*
- reset, *86*
- Reset, *30*
- start, *86*
- Start, *30*
- stop, *86*
- Stop, *30*
- suspend, *87*
- Suspend, *30*
- undefined, *86*
- Undefined, *30*
- UnHold, *30*, *87*
- UnSuspend, *30*, *87*

ET_Modes, *31*
- Maintenance, *31*
- Manual, *31*
- Producting, *31*
- Undefined, *31*

ET_Result, *32*
- Disabled, *32*
- Initializing, *32*
- InvalidStateModelNoAbortedState, *33*
- InvalidStateModelNoCompleteState, *33*
- InvalidStateModelNoExecuteState, *33*
- InvalidStateModelNoHeldState, *33*
- InvalidStateModelNoIdleState, *33*
- InvalidStateModelNoResettingState, *33*
- InvalidStateModelNoStoppedState, *33*
- InvalidStateModelNoSuspendedState, *33*
- ModeChangeRequestRejected, *33*
- NumberOfModesRange, *32*
- Ok, *32*
- PointerAdminInvalid, *32*
- PointerInitModeInvalid, *32*
- PointerStatusInvalid, *32*
- ResettingTimerAndCounter, *32*
- Running, *32*
- StateCurrentRange, *32*
- UnitModeCurrentRange, *32*
- UnitModeRange, *33*
- UnknownResult, *33*

ET_States, *34*
Aborted, *34*
Aborting, *34*
Clearing, *34*
Complete, *34*
Completing, *34*
Execute, *34*
Held, *34*
Holding, *34*
Idle, *34*
Resetting, *34*
Starting, *34*
Stopped, *34*
Stopping, *34*
Suspended, *34*
Suspending, *34*
Undefined, *34*
UnHolding, *34*
UnSuspending, *34*

F

FB_DataManagement, *76*
FB_ModeManager, *79*
FB_VisController, *82*
FC_CheckCmd, *86*
FC_EtResultToString, *88*
FC_GetDateTimeAsArray, *89*
FC_InitStateModelChangeStates, *90*
FC_InitStateModelExistingStates, *93*
FC_SetAlarm, *95*
FC_SetWarning, *97*
FR_<BackgroundFrame>, *101*
FR_<State>, *103*
FR_Alarm, *107*
FR_AlarmHistory, *108*
FR_CurrentModeAndStateTime, *109*
FR_DynStateModel, *110*
FR_ModeAndStateTime, *112*
FR_ProdConsumedCount, *114*
FR_ProdDefectiveCount, *115*
FR_ProdProcessedCount, *116*
FR_StatesDisabled , *119*
FR_StopReason, *118*
FR_Warning, *117*

G

GCL
PackML, *67*
GPL
PackML, *69*

P

PackML
GCL, *67*
GPL, *69*

S

ST_Administration, *39*
ST_Alarm, *54*
ST_Command, *41*
ST_CountDescrip, *55*
ST_Descriptor, *56*
ST_EquipmentInterlock, *57*
ST_Ingredient, *58*
ST_InitAlarm, *45*
ST_Interface, *59*
ST_Product, *60*
ST_StateInfo, *61*
ST_Status, *42*
ST_Timestamp, *62*
ST_UnitModeDefinition, *46*
ST_VisInterface, *48*
States frames, *103*

V

Visualization Frames, *105*

EcoStruxure Machine Expert
PD_ETest
Library Guide

06/2019

EIO0000002884.00

www.schneider-electric.com

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

Safety Information 5
About the Book 9
Part I General Information 13
Chapter 1 Presentation of the Library 15
General Information 15
Part II Global Variables 17
Chapter 2 Global Constant List 19
Global Constant List (GCL) 19
Chapter 3 Global Parameter List 21
Global Parameter List (GPL) 21
Part III Execution 23
Chapter 4 Enumerations 25
ET_TestReturn 25
Chapter 5 Interfaces 27
IF_ErrorLogger 28
IF_ManagedTestObject 29
IF_TestCase 31
IF_TestResource 32
Part IV MemoryLists 33
Chapter 6 Enumerations 35
ET_MeasurandTypes 35
Chapter 7 Function Blocks 37
FB_MeasurandList 37
Part V Using ETEST 39
Chapter 8 Using ETEST 41
Add an ETEST Object 42
Edit Test Cases 43
Use Test Series 45
Use Test Resources 47
Execute Test 48
Record Measured Values 49
Evaluate Test Results 50
Index 53

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

BEFORE YOU BEGIN

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

⚠ WARNING

UNGUARDED EQUIPMENT

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

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

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

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

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

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

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

START-UP AND TEST

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

⚠ WARNING

EQUIPMENT OPERATION HAZARD

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

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

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

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

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

Before energizing equipment:

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

OPERATION AND ADJUSTMENTS

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

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

About the Book

At a Glance

Document Scope

This document describes the PD_ETest library.

The PD_ETest library provides a collection of assistant function blocks for the ETEST framework of the Logic Builder. For further information on the ETEST framework and its use, refer to the ETEST User Guide *(see EcoStruxure Machine Expert, ETEST, User Guide)*.

Validity Note

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

Related Documents

Title of documenation	Reference number
EcoStruxure Machine Expert - ETEST User Guide	*EIO0000002878 (ENG)*; *EIO0000002879 (FRE)*; *EIO0000002880 (GER)*; *EIO0000002881 (ITA)*; *EIO0000002882 (SPA)*; *EIO0000002883 (CHS)*

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

Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

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

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
General Information

Chapter 1
Presentation of the Library

General Information

Introduction

The PD_ETest library provides a collection of assistant function blocks for the ETEST framework of the Logic Builder. For further information on the ETEST framework and its use, refer to the ETEST User Guide *(see EcoStruxure Machine Expert, ETEST, User Guide)*.

When you create a test case in the Logic Builder, the PD_ETest library is automatically integrated in your project. You cannot use ETEST framework without this library.

NOTE: The PD_ETest library is only used in connection with the ETEST framework that is integrated in Logic Builder. The functions and function blocks contained in this library are used implicitly, and are not programmed in the user application.
The POUs of the PD_ETest library can only be accessed from test case objects and resource objects, which have been created in the ETEST framework.
If you attempt to use these functions and functions blocks directly in your application, it can lead to compile errors.

Part II
Global Variables

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
2	Global Constant List	19
3	Global Parameter List	21

Chapter 2
Global Constant List

Global Constant List (GCL)

Overview

Type:	Global constant
Available as of:	V1.0.0.0

Description

The global constant list contains the global constants of the PD_ETest library.

Global Constant

Variable	Data type	Description
`Gc_sLibraryVersion`	STRING[80]	Library version

Chapter 3
Global Parameter List

Global Parameter List (GPL)

Overview

Type:	Global parameter
Available as of:	V1.0.0.0

Description

The global parameter list contains the global parameters of the PD_ETest library. They can be overwritten by project-specific values in the **Library Manager**.

Global Parameters

Variable	Data type	Value	Description
`Gc_uiMaxNumberOfTestCases`	UINT	500	Maximum number of test cases that can be stored.
`Gc_uiMaxNumberOfTcResAssignments`	UINT	800	Maximum number of resources that can be assigned to a test case.
`Gc_uiMaxNumberOfFailedTc`	UINT	300	Maximum number of test cases not successfully completed that can be stored.
`Gc_uiMaxNumberOfResources`	UINT	300	Maximum number of resources that can be stored.
`Gc_uiMaxNumberOfResResAssignments`	UINT	800	Maximum number of test resource assignments to resources that can be stored.
`Gc_uiResourceListSize`	UINT	50	Maximum size of the resources list that can be stored.
`Gc_uiMaxNumberOfMeasurands`	UINT	50	Maximum number of measured values that can be stored.

Part III
Execution

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
4	Enumerations	25
5	Interfaces	27

Chapter 4
Enumerations

ET_TestReturn

Overview

Type:	Enumeration
Available as of:	V1.0.0.0
Default value:	E_Working

Description

The enumeration ET_TestReturn includes the states of the test. It is relevant for internal execution of the test.

Enumeration Elements

Name	Value	Description
E_Working	0	Test case / test sequence is executed.
E_Done	1	Test case / test sequence completed successfully.
E_Failed	2	Test case / test sequence not successful.

Used By

ET_TestReturn is the datatype of the return value of IF_TestCase *(see page 31)*.

Chapter 5
Interfaces

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
IF_ErrorLogger	28
IF_ManagedTestObject	29
IF_TestCase	31
IF_TestResource	32

IF_ErrorLogger

Overview

Type:	Interface
Available as of:	V1.0.0.0

NOTE: This interface type is not intended for use in user logic.

Task

The `IF_ErrorLogger` interface is used to report detected errors.

Functional Description

The methods and properties of this interface are not visible or documented because it is only supposed to be used via the macros (for example, `ASSERT(…)`) of the ETEST framework in Logic Builder.

IF_ManagedTestObject

Overview

Type:	Interface
Available as of:	V1.0.0.0

NOTE: This interface type is not intended for use in user logic.

Task

The IF_ManagedTestObject interface combines joint methods of the objects that have to be prepared and adjusted. It is used for the data inventory in the PD_ETest library.

CleanUp Method

The CleanUp method is used to reinitialize the object(s) of the test cases and test resources so that the object(s) can be re-used.

Interface

Input	Data type	Description
i_xFirstCall	BOOL	TRUE: The method is called for the first time. FALSE: After the first call to the method. Can be used to initialize state machines.
i_ifErrorLogger	IF_ErrorLogger	Interface to report detected errors.

Return value

Data type	Description
ET_TestReturn	Refer to the ET_TestReturn enumeration *(see page 25)*.

`Prepare` Method

The `Prepare` method is used for preparation measures for the tests, such as the initialization of variables or the activation of the position control of an axis.

Interface

Input	Data type	Description
`i_xFirstCall`	BOOL	TRUE: The method is called for the first time. FALSE: After the first call to the method. Can be used to initialize state machines.
`i_rfbResources`	Reference to `FB_Resource List`	List of the resources used.
`i_ifErrorLogger`	`IF_ErrorLogger`	Interface to report detected errors.

Return value

Data type	Description
`ET_TestReturn`	Refer to the `ET_TestReturn` enumeration *(see page 25)*.

IF_TestCase

Overview

Type:	Interface
Available as of:	V1.0.0.0

NOTE: This interface type is not intended for use in user logic.

Task

The IF_TestCase interface is automatically implemented in every test case.

Execute Method

The Execute method contains the actions and verifications of the tests.

Interface

Input	Data type	Description
i_xFirstCall	BOOL	TRUE: The method is called for the first time. FALSE: After the first call to the method. Can be used to initialize state machines.
i_ifErrorLogger	IF_ErrorLogger	Interface to report detected errors.

Output	Data type	Description
q_rProgress	REAL	Used to display the test case progress. Write-enabled parameter.
q_sState	STRING[255]	Textual description of the status of a test case. Write-enabled parameter.

Return value

Data type	Description
ET_TestReturn	Refer to the ET_TestReturn enumeration *(see page 25)*.

Finalize Method

The Finalize method is performed when the Execute method has been completed successfully or not, or when it has been canceled. In contrast to CleanUp, the Finalize method is performed in the same cycle as the Execute method.

IF_TestResource

Overview

Type:	Interface
Available as of:	V1.0.0.0

NOTE: This interface type is not intended for use in user logic.

Task

The `IF_TestResource` interface has to be implemented from a test resource.

It is used to manage test resources.

Used By

`IF_ManagedTestObject` *(see page 29)*

Part IV
MemoryLists

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
6	Enumerations	35
7	Function Blocks	37

Chapter 6
Enumerations

ET_MeasurandTypes

Overview

Type:	Enumeration
Available as of:	V1.0.0.0
Default:	E_NotClassified

Description

The enumeration includes the types of measurand.

Enumeration Elements

Name	Value	Description
E_NotClassified	0	The measured value type is not defined. You can use the input i_sName of the AddMeasurand method to provide information on the measured value.
E_ExecutionTime	1	The measured value is the execution time of the module in nanoseconds. You can use the input i_sName of the AddMeasurand method to provide more information on the time that is measured.
E_SizeOf	2	The measured value is the size of the module in bytes.

Used By

FB_MeasurandList *(see page 37)*

Chapter 7
Function Blocks

FB_MeasurandList

Overview

Type:	Function block
Available as of:	V1.0.0.0

Task

Recording the measured values.

Functional Description

The function block FB_MeasurandList manages the values measured with the AddMeasurand method.

AddMeasurand Method

The AddMeasurand method stores external recorded measured values as a STRING value. For example, the execution times of a function while using specific parameters or the data size of a structure (determined with SIZEOF) can be recorded or stored.

The method can be called at any position within a test case. A test case can provide any number of measured values by calling the method several times. For further information, refer to the chapter *Record Measured Values* *(see page 49)*.

Interface

Input	Data type	Description
i_sModule	STRING[255]	Name of the unit that transfers the measured value. Used to assign the measured value to a library, function block, or function.
i_sName	STRING[255]	Name of the measured value.
i_etType	ET_MeasurandTypes	Type of the measured value. Also refer to the enumeration ET_MeasurandTypes *(see page 35)*. This entry is optional.
i_sValue	STRING[255]	Measured value.

Return value

Data type	Description
UDINT	Indicates the number of stored measured values. The return value is equal to zero if the maximum number of measured values that can be recorded was exceeded and no measured value can be stored anymore. Refer to the parameter Gc_uiMaxNumberOfMeasurands *(see page 21)*.

Part V
Using ETEST

Chapter 8
Using ETEST

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Add an ETEST Object	42
Edit Test Cases	43
Use Test Series	45
Use Test Resources	47
Execute Test	48
Record Measured Values	49
Evaluate Test Results	50

Add an ETEST Object

Add an ETEST Object

To add an ETEST object to your project, proceed as follows:

Step	Action
1	Right-click the **Application** node in the **Tools tree** and execute the **Add Object...** command from the contextual menu.
2	Select the ETEST object you want to add (**Test Case**, **Test Resources**, **Test Series**, **Test Sets**). **Result:** The **ETEST Object** dialog box opens.
3	Rename your object by typing a name in the box **Name**. **NOTE:** Choose a name that complies to the IEC standard. Do not use special characters, leading digits, or spaces within the name. The name must not exceed a length of 32 characters. If you do not rename the object, a name is given by default.
4	Click the **Add** button. **Result:** The selected ETEST object is added to the project and appears as a new node in the **Tools tree**.

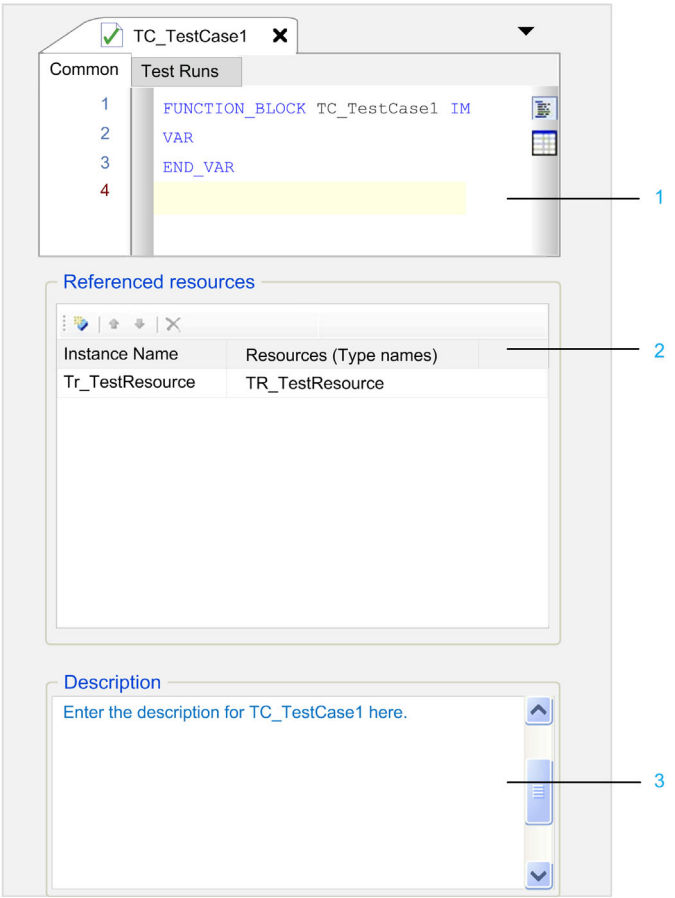
Edit Test Cases

Overview

The structure of test cases is similar to function blocks, but there is no editor available for the implementation section. Only the methods of test cases are implemented. To implement the test case, use the method editor by double-clicking the method in the **Tools tree**.

Common

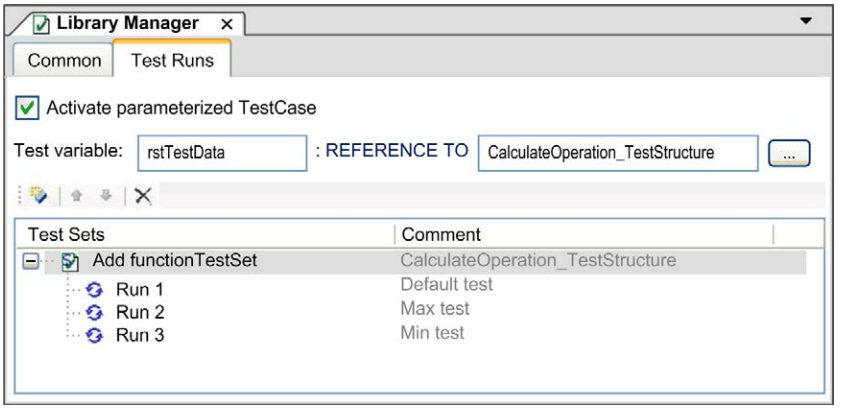
Common tab



1 The declaration editor corresponds to the declaration section of a function block. You can switch between the text view and the table view of the editor using the buttons **Textual** and **Tabular**.
2 The **Referenced resources** area provides integrated test resources.
3 The **Description** area provides space for explanations regarding the test case.

Test Runs

The tab **Test Runs** provides the configuration of test sets.



Button / box	Description
Activate parameterized TestCase	The configured test runs are automatically executed with the test case.
Start Test	Clicking **Start test** from the contextual menu of a single test run, executes the selected test run.
Insert	Clicking **Insert** adds a new **Test Set** entry. With an additional double-click on the test set cell, you can enter a reference to a test set. The existing test set names are suggested as you type. When adding the first test set, the field **REFERENCE TO** is automatically set to the test data type of the newly added **Test set**. **NOTE:** All configured test sets in one test case have to reference the same data type; otherwise a compiler error is detected.
Test Variable	The variable name that can be accessed by the code. The default name is `rstTestData`, but it can be changed.

Navigation

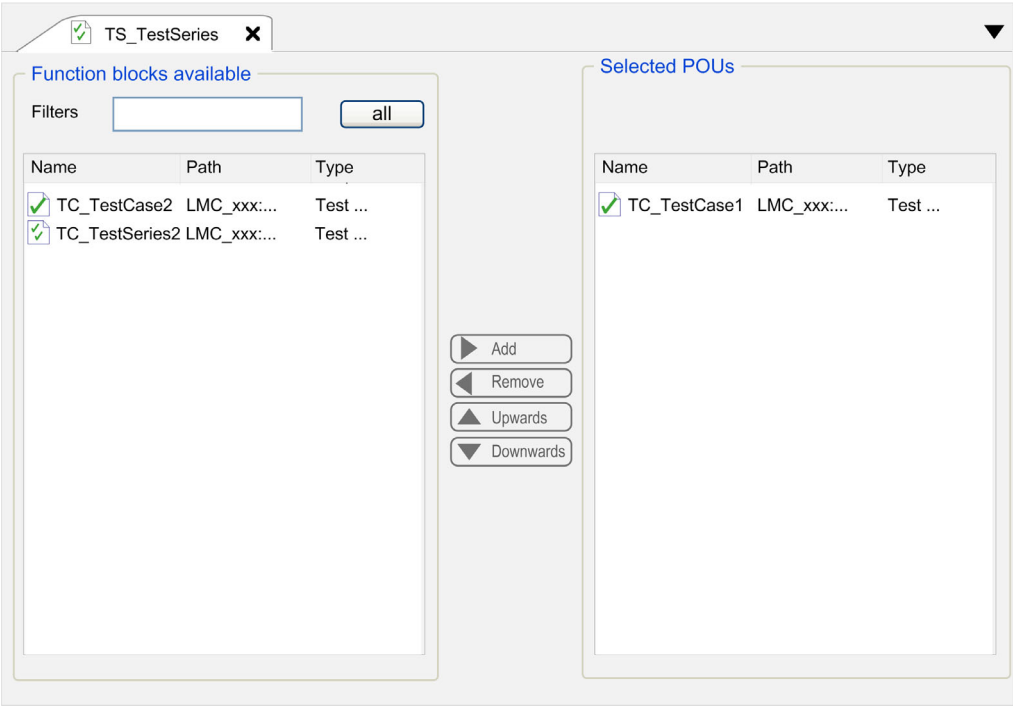
You can use the **Go To Definition…** command from the contextual menu to navigate from a test case to the artifacts used. This function works for both, the configured test data type (IEC-STRUCT), as well as for the referenced test sets. Selecting one of these objects and clicking the **Go to definition…** command opens the corresponding editor for this object.

In addition, the table for referencing test sets also supports standard editing commands.

Use Test Series

Overview

One project comprises several test cases. The **TestSeries** editor allows you to group several test cases into a test series to execute them in a sequence.



The **TestSeries** editor consists of two areas:

Area	Description
Function blocks available	This list includes test cases and test series that are subnodes of the same **Application** node as the open test series. It also lists those test cases and test series that are located in the **Tools tree**. You cannot select a test series as a subnode of itself. Cyclical references between test series are not allowed.
Selected POUs	The selected POUs are displayed. The POUs are the test cases. A test series can select a POU directly only once. Indirectly, POUs can be selected several times. It is thus permissible that test series may embed other test series, and that these embedded test series in turn contain the same test case.

Button / box	Description
Add	Moves a selected test case from the list **Function blocks available** into the list **Selected POUs**.
Remove	Moves a selected test case from the list **Selected POUs** to the list **Function blocks available**.
Upwards	Moves a test case in the area **Selected POUs** upward in the list.
Downwards	Moves a test case in the area **Selected POUs** downward in the list.
Filters all	Enter a text in the **Filters** field to filter the list of **Function blocks available** in accordance with the entry. Click the button **all** to clear the filter.

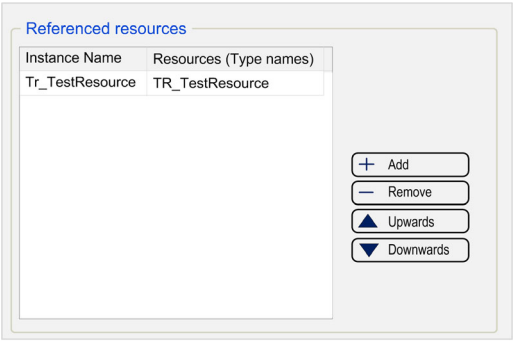
Mouse or Keyboard Action	Description
Double-click	Double-click an entry in the **Function blocks available** list to shift it to the **Selected POUs** list and vice versa.
Delete key	Moves a selected test case from the list **Selected POUs** to the list **Function blocks available**.
Ctrl+A shortcut	Selects all entries of the list.

Use Test Resources

Overview

A test resource encapsulates the access as well as the initialization of hardware and larger data structures used by test cases. Resources are created and processed just as test cases *(see page 43)*. A test case or a resource can bind a resource.

The declaration section for resources (**Referenced resources**) is located in the central area of the editor.



Button	Description
+ Add	Adds a test resource.
- Remove	Removes a test resource.
Upwards	Moves a selected test resource upward within the list.
Downwards	Moves a selected test resource downward within the list.

Mouse or Keyboard Action	Description
Double-click	Double-click an entry in the list of **Referenced resources** to edit it.
Delete key	Removes the test resource.
Ctrl+**A** shortcut	Selects the entries of the **Referenced resources** list.
Ctrl+click or **Shift**+click shortcut	Selects several entries of the **Referenced resources** list.

Each line of the table comprises a referenced resource. The column **Instance name** contains the name of the resource. This name is used if the resource is called from within a test object: A variable of this name is generated, which holds a reference for the resource.

The column **Resources (Type names)** contains the type name of the resource. This is the name of the test resource object which is to be integrated. In the online view, resources are displayed together with the other variables.

Execute Test

Overview

This section describes how to execute test cases and test series *(see page 45)*.

ETEST consists of two main test processes. One process runs within the Logic Builder on the PC. The other process runs on the controller. The controller process is integrated in the application which runs on the controller. Both processes communicate via a communication data structure.

Executing a Test

The following prerequisites must be fulfilled before a test can be executed:

- The application must contain a test object that is referenced in a task.
- The test case or the test series that shall be executed must be selected in the **Tools** tree.
- No other test must be running.

Execute a test as follows:

Step	Action
1	Right-click the node of the test case to be executed in the **Tools** tree. **Result**: The context menu opens.
2	Execute the command **Start test** from the context menu. **Result**: The test case is executed.

After you have started the test, the results view of the test opens. It indicates the progress of the test. You can stop a running test by clicking the **Cancel test run** icon in the toolbar.

NOTE: While a test is running, observe the following:

- Do not disconnect the controller.
- Do not log out.
- Do not stop or reset the application.
- Do not edit the project.

Doing so interrupts the test logic rendering any result invalid.

Record Measured Values

Overview

The example in this chapter illustrates how measured values can be recorded.

To achieve this, the following programming code must be called in a method that is executed within a test.

Example

```
TestManager.fbMeasurandList.AddMeasurand('module', 'name', ET_Measurand
Types.E_NotClassified, 'value');
```

For further information, refer to:

- `FB_MeasurandList` *(see page 37)*
- `ET_MeasurandTypes` *(see page 35)*

Every time a command is executed, a new measured value is recorded.

`TestManager` is the name of **TestManager**-object *(see EcoStruxure Machine Expert, ETEST, User Guide)* of the application.

Evaluate Test Results

Overview

The ETEST framework provides the test results of a completed test and shows the progress of a running test in the **Test results** view.

NOTE: If test cases are built for function blocks contained in libraries, verify the compatibility of any library updates as it concerns your test cases.

Executed Tests

The **Tests runs** list contains the tests that have been executed since Logic Builder has been started.

Buttons	Description
Export	Stores selected test results in XML format. (**.testresult*).
Import	Opens test results stored in XML format.
Print report	Opens an HTML report in the web browser.

Test Results

Click an entry in the **Tests runs** list to open the test in the area **Test results**.

Imported test results are also opened in the area **Test results** and shown in the list of the most recently executed tests. The area **Test results** is automatically opened when a test is started. It shows the result (or the state) of a test.

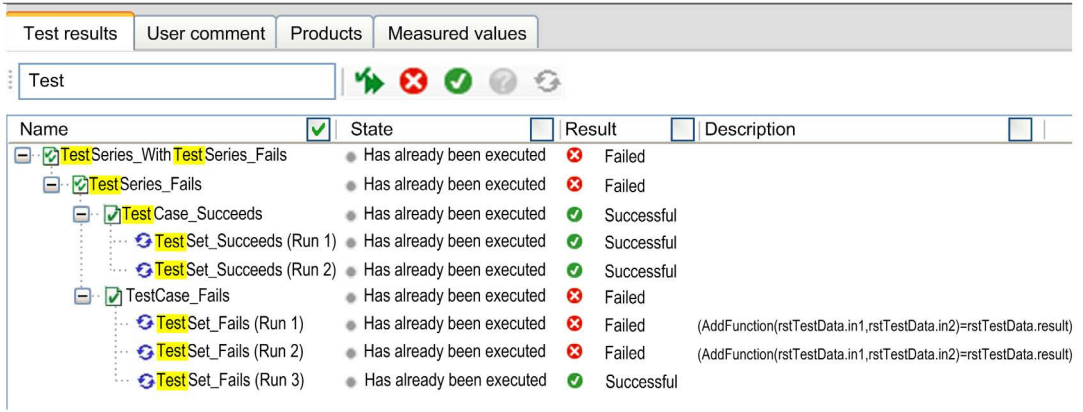
The area **Test results** provides two sections.

- At the top, there is the summary of the test results that contains the name of the executed POU and the total result.
- Below, there is a section with various tabs:

Tab	Description
Test results	Tree view which shows the test series and test cases. It displays the individual results for each node.
User comment	User name and comment. Shows by default the Windows user name. You can add a comment to the **Comment** box. This data is stored together with the test result.
Products	Lists the libraries available in the project during the test.
Measured values	Lists measured values that were recorded during the execution of the test. Used to measure time values and the size of data structures.

Filter

You can filter the entries by a string and the selected column. By default, the **Name** column is selected.



A selected part of the found string (in this example, **Test**) is marked yellow. The list shows the marked entries and its parent nodes.

Test Specific Commands

Command	Description
Run All Tests	Runs the tests that appear in the list.
Run Selected Tests	Runs the selected tests of the list.
Cancel test run	Cancels a running test.
Run Failed Tests	Runs the tests of the list that are indicated as unsuccessful.
Run Passed Tests	Runs the tests of the list that are indicated as successful.
Run Not Run Tests	Runs the not-executed tests of a last test run that was previously canceled.
Repeat Last Run	Repeats a test run that was executed previously from the result view.

Index

A

add
    ETEST objects, *42*

E

edit test cases, *43*
ET_MeasurandTypes, *35*
    E_ExecutionTime, *35*
    E_NotClassified , *35*
    E_SizeOf, *35*
ET_TestReturn, *25*
    E_Done, *25*
    E_Failed, *25*
    E_Working, *25*
ETEST
    add objects, *42*
executing
    test, *48*

F

FB_MeasurandList, *37*

G

GCL (Global Constant List)
    PD_ETest, *19*
GPL (Global Parameter List)
    PD_ETest, *21*

I

IF_ErrorLogger, *28*
IF_ManagedTestObject, *29*
IF_TestCase, *31*
IF_TestResource, *32*
interface
    IF_ErrorLogger, *28*
    IF_ManagedTestObject, *29*
    IF_TestCase, *31*
    IF_TestResource, *32*

M

measured values
    PD_ETest, *37*

P

PD_ETest library
    FB_MeasurandList, *37*
    general information, *15*

R

recording measured values, *49*
results
    ETEST, *50*

T

test
    cases, edit, *43*
    executing, *48*
    managers, recording measured values, *49*
    resources, use, *47*
    results, *50*
    series, use, *45*

U

use
    test resources, *47*
    test series, *45*

EcoStruxure Machine Expert

Getting & Setting Real Time Clock SysTimeRtc and SysTimeCore

Library Guide

06/2019

EIO0000002944.00

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

Safety Information 5
About the Book 7
Chapter 1 General Information 11
General Information 11
Chapter 2 Getting Started 13
Adding the Libraries 14
Get the Controller Date and Time 15
Set the Controller Date and Time 18
Chapter 3 SysTimeCore Library 21
3.1 Functions Descriptions 22
SysTimeGetMs 23
SysTimeGetNs 24
SysTimeGetUs 25
Chapter 4 SysTimeRtc Library 27
4.1 Functions Descriptions 28
SysTimeRtcControl 29
SysTimeRtcGet 30
SysTimeRtcHighResGet 31
SysTimeRtcSet 32
SysTimeRtcHighResSet 33
SysTimeRtcConvertDateToUtc 34
SysTimeRtcConvertDateToHighRes 35
SysTimeRtcConvertUtcToDate 36
SysTimeRtcConvertHighResToDate 37
SysTimeRtcGetTimezone 38
SysTimeRtcSetTimezone 39
SysTimeRtcConvertLocalToUtc 40
SysTimeRtcConvertLocalToHighRes 41
SysTimeRtcConvertUtcToLocal 42
SysTimeRtcConvertHighResToLocal 43
4.2 Data Type Descriptions 44
SYSTIMEDATE 45
RTS_SYSTIMEDATE 45
TimezoneInformation 46
Appendices 47

Appendix A Function and Function Block Representation 49
Differences Between a Function and a Function Block 50
How to Use a Function or a Function Block in IL Language 51
How to Use a Function or a Function Block in ST Language. 55
Glossary 59
Index 61

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

About the Book

At a Glance

Document Scope

This documentation describes the libraries SysTimeRtc and SysTimeCore.

The SysTimeRtc library implements functions for the real time clock (RTC) management of the runtime system.

The SysTimeCore library provides access to high resolution time routines for time calculations.

The following table shows the list of the functions splitted in the two libraries and the controllers that support them:

Function Name	Library Name	PacDrive LMC, LMC078	M241, M251, M258, M262 LMC058	HMISCU
SysTimeGetMs	SysTimeCore	x	x	x
SysTimeGetNs	SysTimeCore	x	x	x
SysTimeGetUs	SysTimeCore	x	x	x
SysTimeRtcConvertDateToHighRes	SysTimeRtc	x	x	-
SysTimeRtcConvertHighResToDate	SysTimeRtc	x	x	-
SysTimeRtcHighResGet	SysTimeRtc	x	x	-
SysTimeRtcHighResSet	SysTimeRtc	x	x	-
SysTimeRtcControl	SysTimeRtc	x	-	x
SysTimeRtcConvertDateToUtc	SysTimeRtc	x	x	x
SysTimeRtcConvertUtcToDate	SysTimeRtc	x	x	x
SysTimeRtcConvertLocalToUtc	SysTimeRtc	x	x	-
SysTimeRtcConvertUtcToLocal	SysTimeRtc	x	x	-
SysTimeRtcGet	SysTimeRtc	x	x	x
SysTimeRtcSet	SysTimeRtc	x	x	x
SysTimeRtcConvertHighResToLocal	SysTimeRtc	x	x	-
SysTimeRtcConvertLocalToHighRes	SysTimeRtc	x	x	-
SysTimeRtcGetTimezone	SysTimeRtc	x	x	x
SysTimeRtcSetTimezone	SysTimeRtc	x	x	x
x Supported **-** Not supported				

Validity Note

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

Related Documents

Document title	Reference
Modicon M241 Logic Controller - System Functions and Variables - PLCSystem Library Guide	*EIO0000001438 (ENG)*; *EIO0000001439 (FRE)*; *EIO0000001440 (GER)*; *EIO0000001441 (SPA)*; *EIO0000001442 (ITA)*; *EIO0000001443 (CHS)*
Modicon M251 Logic Controller - System Functions and Variables - PLCSystem Library Guide	*EIO0000001468 (ENG)*; *EIO0000001469 (FRE)*; *EIO0000001470 (GER)*; *EIO0000001471 (SPA)*; *EIO0000001472 (ITA)*; *EIO0000001473 (CHS)*
Modicon M258 Logic Controller - System Functions and Variables - PLCSystem Library Guide	*EIO0000000584 (ENG)*; *EIO0000000585 (FRE)*; *EIO0000000586 (GER)*; *EIO0000000587 (ITA)*; *EIO0000000588 (SPA)*; *EIO0000000589 (CHS)*
Modicon LMC058 Motion Controller - System Functions and Variables - PLCSystem Library Guide	*EIO0000000566 (ENG)*; *EIO0000000567 (FRE)*; *EIO0000000568 (GER)*; *EIO0000000569 (ITA)*; *EIO0000000570 (SPA)*; *EIO0000000571 (CHS)*
Modicon LMC078 Motion Controller - System Functions and Variables - PLCSystem Library Guide	*EIO0000001917 (ENG)*; *EIO0000001918 (FRE)*; *EIO0000001919 (GER)*; *EIO0000001920 (SPA)*; *EIO0000001921 (ITA)*; *EIO0000001922 (CHS)*

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

Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

⚠ 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.

Chapter 1
General Information

General Information

Overview

Depending on the runtime system, the real time clock (RTC) of the controller is provided in standard or high resolution time stamp. That is, the time stamp represents the seconds (standard resolution) or milliseconds (high resolution) since January 1st, 1970 00:00:00. It is also known as UNIX format.

The RTC in standard resolution is supported for all EcoStruxure Machine Expert controllers. Refer to the table *(see page 7)* in the Document Scope chapter to verify for which controller supports the high resolution RTC.

Chapter 2
Getting Started

Overview

This chapter provides essential information to start using the `SysTime` functions.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Adding the Libraries	14
Get the Controller Date and Time	15
Set the Controller Date and Time	18

Adding the Libraries

Procedure

To have access to the functions, it is necessary to manually add one or both libraries:

Step	Action
1	Double-click the **Library Manager** node of the controller **Application** node in the **Tools tree**.
2	Click **Add Library**.
3	Browse to **System → SysLibs**, visible when the **Company** filter is set to **System** or **All Company**. **Result:** The System libraries list is displayed.
4	Select the SysTimeRtc or SysTimeCore library and press **OK**. **Result:** The library is added to the **Library Manager** list.

Get the Controller Date and Time

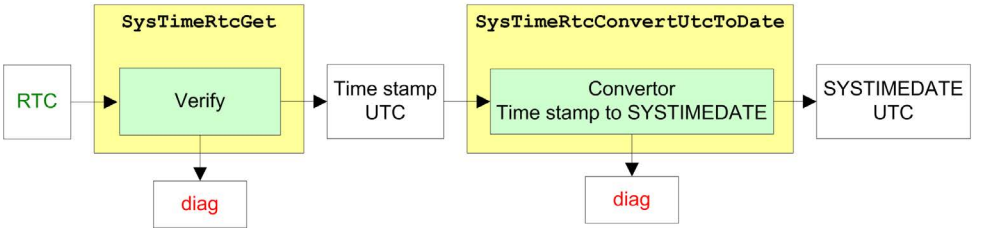
Overview

To get the RTC of the controller in a structured and ergonomic format, you have to use 2 different functions.

1. Read the RTC, using the functions `SysTimeRtcGet` *(see page 30)* or `SysTimeRtcHighResGet` *(see page 31)*.
2. Convert the time stamp in UNIX format to the SYSTIMEDATE format with the use of the function `SysTimeRtcConvertUtcToDate` *(see page 36)* or `SysTimeRtcConvertHighResToDate` *(see page 37)*.

NOTE: Due to the fact that only the UTC (Coordinated Universal Time) time is globally unique, on most systems only the UTC time is stored and processed.

Principle Diagram - Get the RTC of the Controller in Standard Resolution



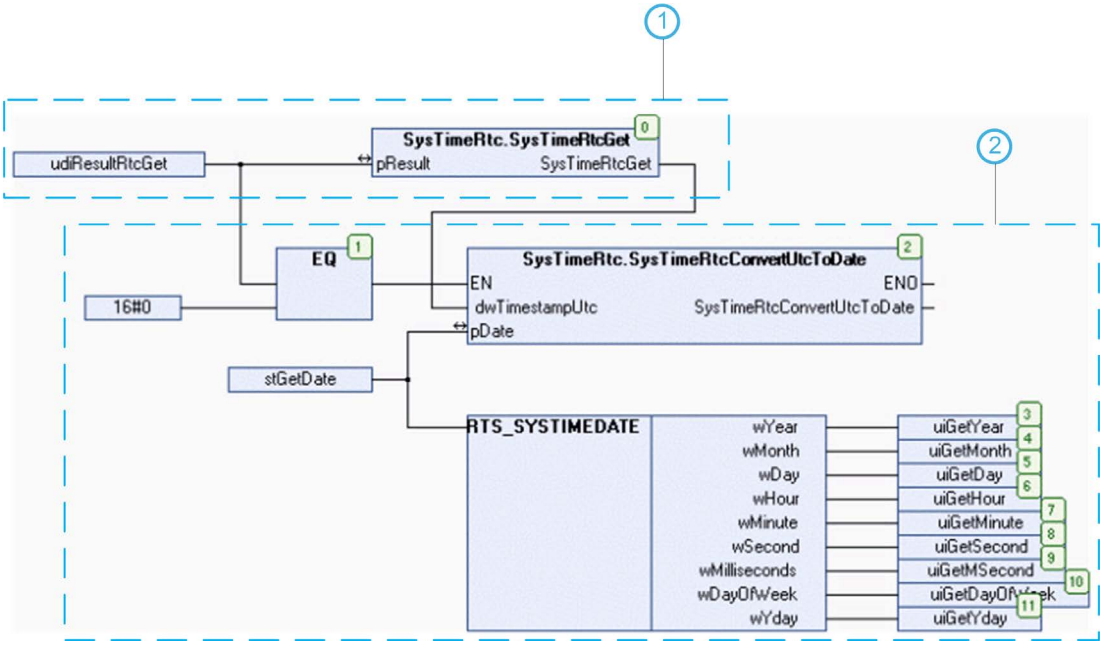
Example

This program example can be used to get the controller date and time.

Variable declaration:

```
VAR
   uidResultRtcGet: UDINT
   stGetDate: SysTimeRtc.RTS_SYSTIMEDATE;
   uiGetYear: UINT;
   uiGetMonth: UINT;
   uiGetDay: UINT;
   uiGetHour: UINT;
   uiGetMinute: UINT;
   uiGetSecond: UINT;
   uiGetMSecond: UINT;
   uiGetDayOfWeek: UINT;
   uiGetYday: UINT;
   uidResultConvertToDate: UDINT;
END_VAR
```

POU program:



1 Get the RTC of the controller as time stamp value.
2 Convert the time stamp value in SYSTIMEDATE format.

Set the Controller Date and Time

Overview

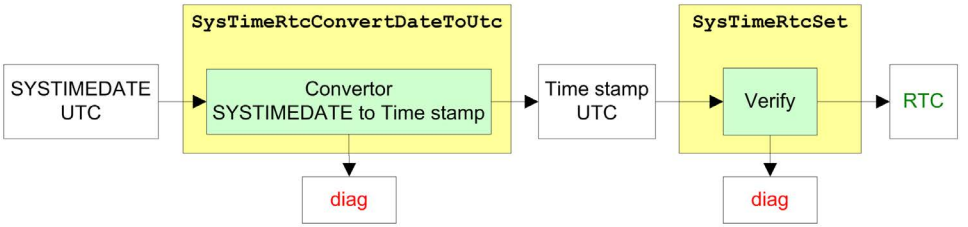
To set the RTC of the controller based on a structured and ergonomic format, you have to use 2 different functions.

1. Convert the SYSTIMEDATE format to the time stamp in UNIX format with the use of the function `SysTimeRtcConvertDateToUtc` *(see page 34)* or `SysTimeRtcConvertDateToHighRes` *(see page 35)*.
2. Write the RTC, using the functions `SysTimeRtcSet` *(see page 32)* or `SysTimeRtcHighResSet` *(see page 33)*.

NOTE: Some controllers support a function for a weekly automatic correction of the real time clock. The name of this function is `SetRTCDrift`. The use of this function could be an alternative to using the `SysTimeRtcSet` function for the continuously readjustment of the RTC. Refer to the PLCSystem Library Guide *(see page 8)* of your controller to verify whether the function is supported and to get further information about this function.

NOTE: Due to the fact that only the UTC (Coordinated Universal Time) time is globally unique, on most systems only the UTC time is stored and processed.

Principle Diagram - Set the RTC of the Controller in Standard Resolution



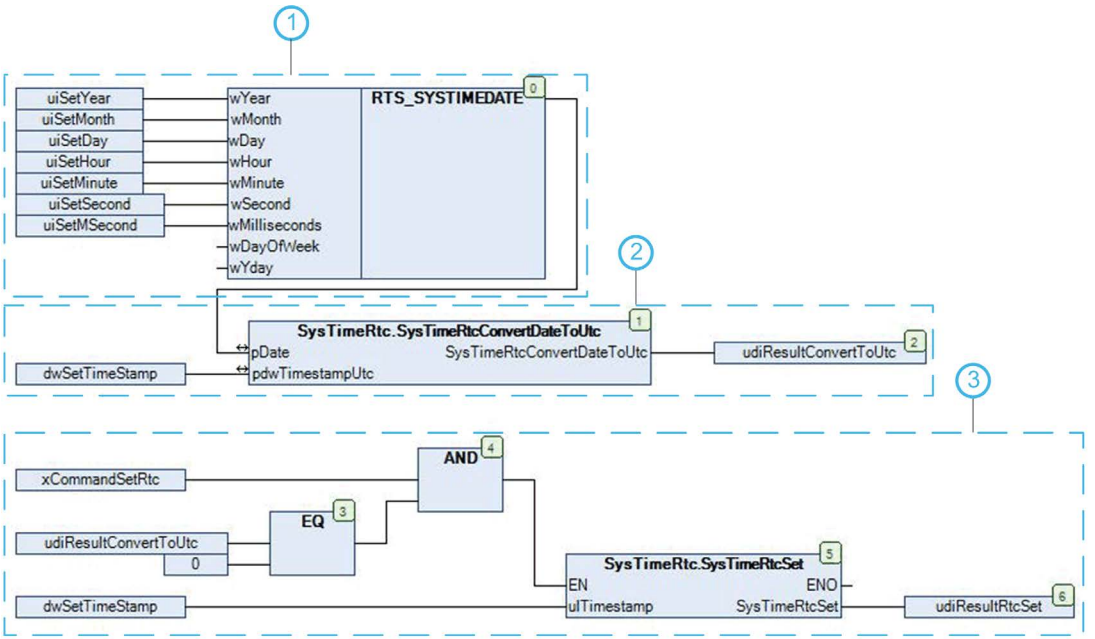
Example

This program example can be used to set the controller real time clock with a user date and time.

Variable declaration:

```
VAR
   uiSetYear: UINT;
   uiSetMonth: UINT;
   uiSetDay: UINT;
   uiSetHour: UINT;
   uiSetMinute: UINT;
   uiSetSecond: UINT;
   uiSetMSecond: UINT
   udiResultConvertToUtc: UDINT;
   dwSetTimeStamp: DWORD;
   xCommandSetRtc: BOOL;
   uidResultRtcSet: UDINT;
END_VAR
```

POU program:



1 Assign the date and time parameter to the structure.
2 Convert the SYSTIMEDATE format to a time stamp value.
3 Set the controller RTC with new time stamp if `xCommandSetRtc` = TRUE and conversion was successful.

Chapter 3
SysTimeCore Library

Section 3.1
Functions Descriptions

Overview

This section describes the functions of the SysTimeCore library.

What Is in This Section?

This section contains the following topics:

Topic	Page
SysTimeGetMs	23
SysTimeGetNs	24
SysTimeGetUs	25

SysTimeGetMs

Function Description

Returns a monotonic rising millisecond counter. This value can be used for timeout and relative time measurements. The counter is reset at each boot-up of the controller.

NOTE: The real time clock does not influence this counter.

Graphical Representation



I/O Variables Description

Output	Type	Description
SysTimeGetMs	UDINT	Value of the millisecond counter, which is reset on boot-up.

SysTimeGetNs

Function Description

Returns a monotonic rising nanosecond counter. This value can be used for timeout and high resolution time measurements.

The counter is reset at each boot-up of the controller.

NOTE: The real time clock does not influence this counter.

Graphical Representation



I/O Variables Description

Output	Type	Description
SysTimeGetNs	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

Input/Output	Type	Description
pUsTime	SYSTIME	Value of the nanosecond counter, which is reset on boot-up.

NOTE: SYSTIME is an alias type based on the data type ULINT.

SysTimeGetUs

Function Description

Returns a monotonic rising microsecond counter. This value can be used for timeout and high resolution time measurements.

The counter is reset at each boot-up of the controller.

NOTE: The real time clock does not influence this counter.

Graphical Representation



I/O Variables Description

Output	Type	Description
SysTimeGetUs	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

Input/Output	Type	Description
pUsTime	SYSTIME	Value of the microsecond counter, which is reset on boot-up.

NOTE: SYSTIME is an alias type based on the data type ULINT.

Chapter 4
SysTimeRtc Library

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
4.1	Functions Descriptions	28
4.2	Data Type Descriptions	44

Section 4.1
Functions Descriptions

Overview

This section describes the functions of the SysTimeRtc library.

The function for the real time clock handle two different time types:

UTC time: coordinated universal time; has replaced the Greenwich Mean Time. The time zones are given as positive or negative deviation from UTC. For example, UTC+1 corresponds to the Central European Time (CET) and UTC+2 corresponds to the Central European Summer Time (CEST).

Local time: Local time, including daylight saving time (summer time) and time zone shift.

NOTE: Due to the fact that only the UTC (Coordinated Universal Time) time is globally unique, on most systems only the UTC time is stored and processed.

What Is in This Section?

This section contains the following topics:

Topic	Page
SysTimeRtcControl	29
SysTimeRtcGet	30
SysTimeRtcHighResGet	31
SysTimeRtcSet	32
SysTimeRtcHighResSet	33
SysTimeRtcConvertDateToUtc	34
SysTimeRtcConvertDateToHighRes	35
SysTimeRtcConvertUtcToDate	36
SysTimeRtcConvertHighResToDate	37
SysTimeRtcGetTimezone	38
SysTimeRtcSetTimezone	39
SysTimeRtcConvertLocalToUtc	40
SysTimeRtcConvertLocalToHighRes	41
SysTimeRtcConvertUtcToLocal	42
SysTimeRtcConvertHighResToLocal	43

SysTimeRtcControl

Function Description

This function is used to read the hardware status information related to the real time clock (RTC).

This function is not available for all supported controllers. Consult the *Programming Guide* specific to your controller for further information.

Graphical Representation



I/O Variables Description

Input	Type	Description
iControlTag	DINT	0 = to verify the battery of the RTC 1 = to verify the hour format of the RTC

Input/Output	Type	Description
pdiControlResult	DINT	Result of the verification. Hour format verification: 0 = 12-hour format 1 = 24-hour format Battery verification: 0 = indicates that the battery needs to be replaced 1 = battery is ok

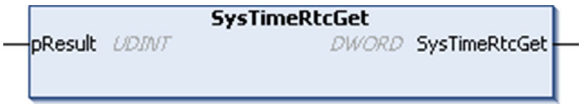
Output	Type	Description
SysTimeRtcControl	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

SysTimeRtcGet

Function Description

This function is used to read the value of the real time clock (RTC) of the controller. The RTC is provided as time stamp which indicates the number of seconds since January 1st, 1970 00:00:00.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
pResult	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

Output	Type	Description
SysTimeRtcGet	DWORD	RTC of the controller (number of seconds since January 1st, 1970 00:00:00)

NOTE: An example using this function is provided in this document. *(see page 16)*

SysTimeRtcHighResGet

Function Description

This function is used to read the value of the real time clock (RTC) of the controller. The RTC is provided as high resolution time stamp which indicates the number of milliseconds since January 1st, 1970 00:00:00:000.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
pTimestamp	SYSTIME	Time in milliseconds since January 1st, 1970 00:00:00:000

Output	Type	Description
SysTimeRtcHighResGet	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: SYSTIME is an alias type based on the data type ULINT.

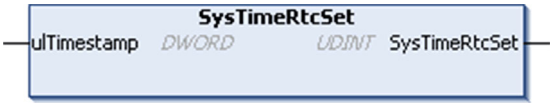
SysTimeRtcSet

Function Description

This function is used to set the real time clock of the controller by a provided time stamp which indicates the numbers of seconds since January 1st, 1970 00:00:00.

NOTE: Setting the RTC of the controller generates entries into the controller log file. Therefore, for automatic adjustments, do not use this function more than once a day.

Graphical Representation



I/O Variables Description

Input	Type	Description
ulTimestamp	DWORD	Time stamp value (number of seconds since January 1st, 1970 00:00:00)

Output	Type	Description
SysTimeRtcSet	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: An example using this function is provided in this document. *(see page 19)*

SysTimeRtcHighResSet

Function Description

This function is used to set the real time clock (RTC) of the controller by a provided high resolution time stamp value which indicates the number of milliseconds since January 1st, 1970 00:00:00:000.

NOTE: Setting the RTC of the controller generates entries into the controller log file. Therefore, for automatic adjustments, do not use this function more than once a day.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
pTimestamp	SYSTIME	Time stamp value (number in milliseconds since January 1st, 1970 00:00:00:000).

Output	Type	Description
SysTimeRtcHighResSet	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

SysTimeRtcConvertDateToUtc

Function Description

This function converts a date and time in SYSTIMEDATE format *(see page 45)* into the corresponding time stamp value. The time stamp indicates the number of seconds since January 1st, 1970 00:00:00.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
pDate	SYSTIMEDATE *(see page 45)*	Date and time to be converted.
pdwTimestampUtc	DWORD	Time stamp calculated from pDate.

Output	Type	Description
SysTimeRtcConvertDateToUtc	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: An example using this function is provided in this document. *(see page 19)*

SysTimeRtcConvertDateToHighRes

Function Description

This function converts a date and time in SYSTIMEDATE *(see page 45)* format into the corresponding high resolution time stamp value. The time stamp indicates the number of milliseconds since January 1st, 1970 00:00:00:000.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
pDate	SYSTIMEDATE *(see page 45)*	The time to be converted.
pTimestamp	SYSTIME	Time stamp calculated from pDate.

Output	Type	Description
SysTimeRtcConvertDateToHighRes	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: SYSTIME is an alias type based on the data type ULINT.

SysTimeRtcConvertUtcToDate

Function Description

This function converts a time stamp value into the corresponding date and time in SYSTIMEDATE format. *(see page 45)* The time stamp indicates the number of seconds since January 1st, 1970 00:00:00.

Graphical Representation



I/O Variables Description

Input	Type	Description
dwTimestampUtc	DWORD	Time stamp to be converted.

Input/Output	Type	Description
pDate	SYSTIMEDATE *(see page 45)*	Date and time calculated from input value.

Output	Type	Description
SysTimeRtcConvertUtcToDate	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: An example using this function is provided in this document. *(see page 16)*

SysTimeRtcConvertHighResToDate

Function Description

This function converts a high resolution time stamp value into the corresponding date and time in SYSTIMEDATE *(see page 45)* format. The time stamp indicates the number of milliseconds since January 1st, 1970 00:00:00:000.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
pTimestamp	SYSTIME	Time stamp to be converted.
pDate	SYSTIMEDATE *(see page 45)*	Date and time calculated from the input value.

Output	Type	Description
SysTimeRtcConvertHighResToDate	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: SYSTIME is an alias type based on the data type ULINT.

SysTimeRtcGetTimezone

Function Description

This function is used to read the timezone settings of the controller.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
pTimezone	TimezoneInformation *(see page 46)*	Timezone settings of the controller.

Output	Type	Description
SysTimeRtcGetTimezone	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

SysTimeRtcSetTimezone

Function Description

This function is used to set the specified timezone settings.

NOTE: The execution of the function `SysTimeRtcSetTimezone` may take several hundred milliseconds. The increased execution time is caused by storing the `TimezoneInformation` parameter to a configuration file in the controller.

To help to avoid blocking other tasks by executing this function, implement one of the following measures:

- Use the asynchronous mechanism provided by the AsyncManager library *(see EcoStruxure Machine Expert, AsyncManager, Library Guide)* to outsource the function call to an external task.
- Implement the function call in a separate task with suitable task configuration.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
`pTimezone`	TimezoneInformation *(see page 46)*	Timezone settings to be set for the controller.

Output	Type	Description
`SysTimeRtcSetTimezone`	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

SysTimeRtcConvertLocalToUtc

Function Description

This function calculates the UTC (Coordinated Universal Time) from a local time considering the timezone setting of the runtime system. Both, the UTC and the local time stamp indicate the number of seconds since January 1st, 1970 00:00:00.

Graphical Representation



I/O Variables Description

Input	Type	Description
dwTimestampLocal	DWORD	Local time stamp

Input/Output	Type	Description
pdwTimestampUTC	DWORD	UTC calculated from the input.

Output	Type	Description
SysTimeRtcConvertLocalToUtc	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

SysTimeRtcConvertLocalToHighRes

Function Description

This function converts a date and time in SYSTIMEDATE *(see page 45)* format into the corresponding high resolution time stamp value considering the timezone settings of the runtime system.

The value `pDate` represents the local time and will be converted to the UTC value, which is provided on `pTimestamp`. The value `pTimestamp` indicates the number of milliseconds since January 1st, 1970 00:00:00:000.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
`pDate`	SYSTIMEDATE *(see page 45)*	Local time to be converted.
`pTimestamp`	SYSTIME	Time stamp calculated from `pDate`.

Output	Type	Description
`SysTimeRtcConvertLocalToHighRes`	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: SYSTIME is an alias type based on the data type ULINT.

SysTimeRtcConvertUtcToLocal

Function Description

This function calculates the local time from the UTC (Coordinated Universal Time) considering the time zone setting of the runtime system. Both, the UTC and the local time stamp indicate the number of seconds since January 1st, 1970 00:00:00.

Graphical Representation



I/O Variables Description

Input	Type	Description
dwTimestampUtc	DWORD	UTC time stamp

Input/Output	Type	Description
pdwTimestampLocal	DWORD	Local time stamp calculated from the input.

Output	Type	Description
SysTimeRtcConvertUtcToLocal	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

SysTimeRtcConvertHighResToLocal

Function Description

This function converts a high resolution time stamp value into the corresponding date and time in SYSTIMEDATE *(see page 45)* format considering the timezone settings of the runtime system. The value `pTimestamp` represents the UTC and will be converted to the local time, which is provided on `pDate`.

Graphical Representation



I/O Variables Description

Input/Output	Type	Description
`pTimestamp`	SYSTIME	Time stamp to be converted.
`pDate`	SYSTIMEDATE *(see page 45)*	Local time calculated from `pTimestamp`.

Output	Type	Description
`SysTimeRtcConvertHighResToLocal`	UDINT	Runtime system error code (refer to CmpErrors.library): 0 = no error detected

NOTE: SYSTIME is an alias type based on the data type ULINT.

Section 4.2
Data Type Descriptions

What Is in This Section?

This section contains the following topics:

Topic	Page
SYSTIMEDATE	45
RTS_SYSTIMEDATE	45
TimezoneInformation	46

SYSTIMEDATE

Data Type Description

The data type SYSTIMEDATE is an alias type based on the data type RTS_SYSTIMEDATE.

RTS_SYSTIMEDATE

Data Type Description

The data type RTS_SYSTIMEDATE is a structure which implements the elements needed to represent a high resolution date and time value.

This structure contains detailed information on the date and time, presented in a readable format (in contrast to the time stamp).

Name	Type	Description
wYear	UINT	Year
wMonth	UINT	Month, values 1 to12 (1=January, 12=December)
wDay	UINT	Day of month, values 1 to 31
wHour	UINT	Hours of current day, values 0 to 23
wMinute	UINT	Minutes of current hour, values 0 to 59
wSecond	UINT	Seconds of current minute, values 0 to 59
wMilliseconds	UINT	Milliseconds of current second, values 0 to 999
wDayOfWeek	UINT	Day of the week, values 1 to 7 (1=Monday, 7=Sunday)
wYday	UINT	Day of the year, values 1 to 366 (1=January 1st, 365 or 366=December 31st)

TimezoneInformation

Data Type Description

This information describes a local timezone with standard- and daylight-saving-time (also known as summer and winter time).

Name	Type	Description	Unit
`ulStandardDate`	UDINT	Date to switch to standard time (from summer to winter time).	Number of seconds since January 1st, 1970 00:00:00.
`ulDaylightDate`	UDINT	Date to switch to daylight saving time (from winter to summer time).	
`szStandardName`	STRING (32)	Standard name of the timezone (winter time).	String with maximum 32 characters
`szDaylightName`	STRING (32)	Daylight saving time name (summer time).	String with maximum 32 characters
`iBias`	INT	UTC = local time + bias	minutes
`iStandardBias`	INT	Additional offset at standard time (winter time).	minutes
`iDayligthBias`	INT	Additional offset at daylight saving time (summer time).	minutes

Appendices

Appendix A
Function and Function Block Representation

Overview

Each function can be represented in the following languages:

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

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

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Differences Between a Function and a Function Block	50
How to Use a Function or a Function Block in IL Language	51
How to Use a Function or a Function Block in ST Language	55

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.

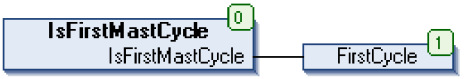
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

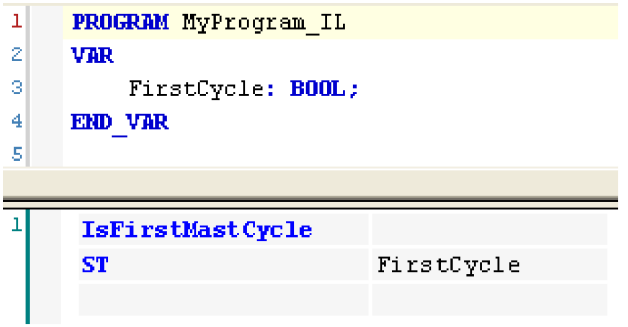
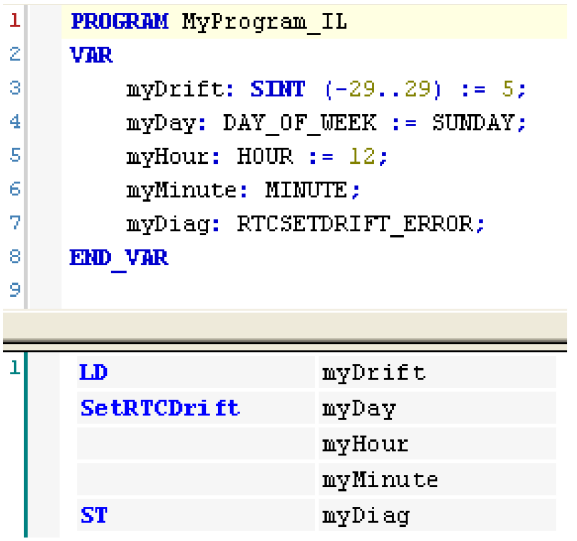
This procedure describes how to insert a function in IL language:

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 EcoStruxure Machine Expert, 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`	
IL example of a function with input parameters: `SetRTCDrift`	

Using a Function Block in IL Language

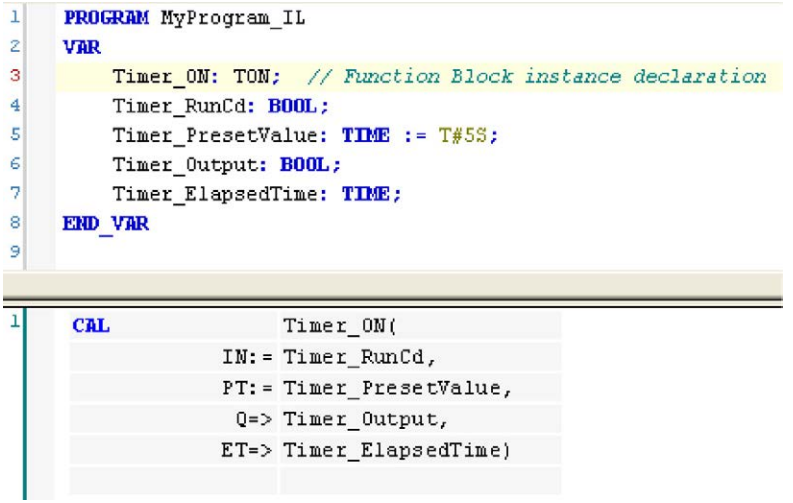
This procedure describes how to insert a function block in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see EcoStruxure Machine Expert, 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 EcoStruxure Machine Expert, 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

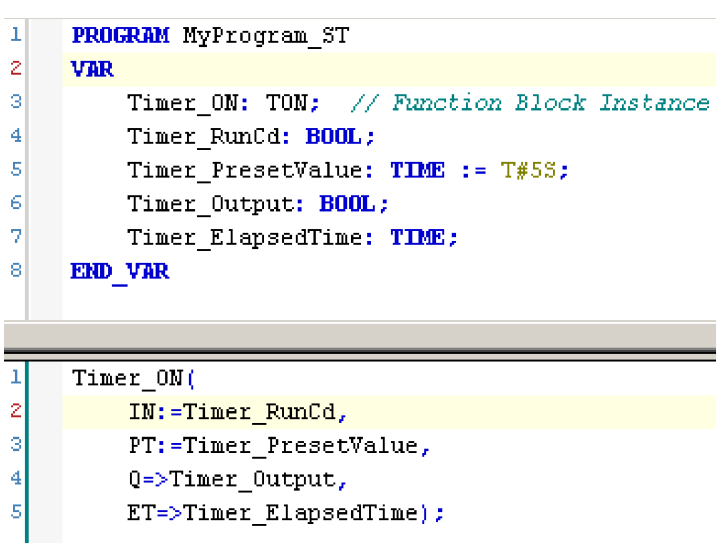
This procedure describes how to insert a function block in ST language:

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 EcoStruxure Machine Expert, 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);
```

Glossary

B

byte

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

C

CFC

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

F

FB

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

function block diagram

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

I

IL

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

INT

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

L

LD

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

P

POU

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

S

ST

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

U

UTC

(*universal time coordinated*) The primary time standard by which the world regulates clocks and time.

V

variable

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

Index

C

converting
- SysTimeRtcConvertDateToUtc, *34*
- SysTimeRtcConvertUtcToDate, *36*

D

data and time of the controller
- getting, *15*

data types
- RTS_SYSTIMEDATE, *45*
- SYSTIMEDATE, *45*
- SysTimeRtc, *44*
- TimezoneInformation, *46*

date
- SysTimeRtcConvertDateToUtc, *34*
- SysTimeRtcConvertUtcToDate, *36*

date and time of the controller
- setting, *18*

E

examples
- getting the controller data and time, *16*
- setting data and time of the controller, *19*

F

functions
- differences between a function and a function block, *50*
- how to use a function or a function block in IL language, *51*
- how to use a function or a function block in ST language, *55*
- SysTimeCore, *22*
- SysTimeRtc, *28*

G

getting controller RTC, *15*
- examples, *16*

H

high resolution time stamp
- SysTimeRtc, *28*

R

real time clock
- SysTimeRtcControl, *29*
- SysTimeRtcConvertDateToHighRes, *35*
- SysTimeRtcConvertHighResToDate, *37*
- SysTimeRtcConvertHighResToLocal, *43*
- SysTimeRtcConvertLocalToHighRes, *41*
- SysTimeRtcGet, *30*
- SysTimeRtcGetTimezone, *38*
- SysTimeRtcHighResGet, *31*
- SysTimeRtcHighResSet, *33*
- SysTimeRtcSet, *32*
- SysTimeRtcSetTimezone, *39*

RTC
- SysTimeRtcControl, *29*
- SysTimeRtcConvertDateToHighRes, *35*
- SysTimeRtcConvertHighResToDate, *37*
- SysTimeRtcConvertHighResToLocal, *43*
- SysTimeRtcConvertLocalToHighRes, *41*
- SysTimeRtcGet, *30*
- SysTimeRtcHighResGet, *31*
- SysTimeRtcHighResSet, *33*

RTC (real time clock), *13*
- SysTimeRtcGetTimezone, *38*
- SysTimeRtcSet, *32*
- SysTimeRtcSetTimezone, *39*

RTS_SYSTIMEDATE, *45*

S

setting controller RTC, *18*
examples, *19*
standard resolution time stamp
SysTimeRtc, *28*
SysTime functions
SysTimeCore library, *13*
SysTimeRtc library, *13*
SysTimeCore
functions, *22*
general information, *11*
SysTimeGetMs, *23*
SysTimeGetNs, *24*
SysTimeGetUs, *25*
time meters, *22*
SYSTIMEDATE, *45*
SysTimeGetMs, *23*
SysTimeGetNs, *24*
SysTimeGetUs, *25*
SysTimeRtc
functions, *28*
general information, *11*
RTS_SYSTIMEDATE, *45*
SYSTIMEDATE, *45*
SysTimeRtcControl, *29*
SysTimeRtcConvertDateToHighRes, *35*
SysTimeRtcConvertDateToUtc, *34*
SysTimeRtcConvertHighResToDate, *37*
SysTimeRtcConvertHighResToLocal, *43*
SysTimeRtcConvertLocalToHighRes, *41*
SysTimeRtcConvertUtcToDate, *36*
SysTimeRtcConvertUtcToLocal, *42*
SysTimeRtcGet, *30*
SysTimeRtcGetTimezone, *38*
SysTimeRtcHighResGet, *31*
SysTimeRtcHighResSet, *33*
SysTimeRtcSet, *32*
SysTimeRtcSetTimezone, *39*
TimezoneInformation, *46*
SysTimeRtcControl, *29*
SysTimeRtcConvertDateToHighRes, *35*
SysTimeRtcConvertDateToUtc, *34*
SysTimeRtcConvertHighResToDate, *37*
SysTimeRtcConvertHighResToLocal, *43*
SysTimeRtcConvertLocalToHighRes, *41*
SysTimeRtcConvertLocalToUtc, *40*
SysTimeRtcConvertLocalToUtc, *40*
SysTimeRtcConvertUtcToDate, *36*
SysTimeRtcConvertUtcToLocal, *42*
SysTimeRtcGet, *30*
SysTimeRtcGetTimezone, *38*
SysTimeRtcHighResGet, *31*
SysTimeRtcHighResSet, *33*
SysTimeRtcSet, *32*
SysTimeRtcSetTimezone, *39*

T

time
SysTimeGetMs, *23*
SysTimeGetNs, *24*
SysTimeGetUs, *25*
time meters
SysTimeCore, *22*
time stamp
SysTimeRtc, *28*
SysTimeRtcConvertDateToUtc, *34*
SysTimeRtcConvertUtcToDate, *36*
TimezoneInformation, *46*

U

universal time coordinated (UTC)
SysTimeRtcConvertLocalToUtc, *40*
SysTimeRtcConvertUtcToLocal, *42*
UTC
SysTimeRtcConvertLocalToUtc, *40*
SysTimeRtcConvertUtcToDate, *36*
SysTimeRtcConvertUtcToLocal, *42*
UTC (universal time coordinated), *13*

EcoStruxure Machine Expert
Miscellaneous Functions Toolbox

Library Guide

06/2019

EIO0000000096.07

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

Safety Information 11
About the Book 15
Part I Function Requirements 19
Chapter 1 Function Requirements 21
Toolbox Function Library Requirements 21
Part II Bit Functions 23
Chapter 2 Bit Organization 25
Bit Organization for DWORD 25
Chapter 3 SetBitTo: Setting/Resetting One Bit 27
`SetBitTo` Function 27
Chapter 4 TestBit: Testing One Bit 29
`TestBit` Function 29
Part III Closed Loop Functions 31
Chapter 5 FB_2points: Closed Loop 2 Point Transfer Element 33
`FB_2points` Function Block 34
Operating Modes 34
Input Pin Description 37
Structure Used 37
Output Pin Description 38
Chapter 6 FB_3points: Closed Loop 3 Points Transfer Element . . . 39
`FB_3points` Function Block 40
Operating Modes 41
Input Pin Description 43
Structure Used 43
Output Pin Description 44
Chapter 7 FB_3points_Ext: Closed Loop Extension For 3 Points Transfer Element 45
`FB_3points_Ext` Function Block 46
Operating Modes 47
Input Pin Description 50
Structure Used 50
Output Pin Description 51

Chapter 8 FB_P: Control Loop With Proportional Only Algorithm . . . 53
FB_P Function Block . . . 54
Operation Modes . . . 55
Input Pin Description . . . 57
Output Pin Description . . . 58
Chapter 9 FB_PI: Proportional and Integration Process Control. . . . 59
FB_PI Function Block . . . 60
Input Pin Description . . . 62
Structure Used . . . 64
Output Pin Description . . . 66
Chapter 10 FB_PID: Manual Mode Operation Process Control 67
FB_PID Function Block . . . 68
Input Pin Description . . . 72
Structure Used . . . 74
Output Pin Description . . . 76
Instantiation and Usage Example . . . 78
Chapter 11 FB_PI_PID: Cascaded PI_PID Control Loop 85
FB_PI_PID Function Block . . . 86
Operation Modes . . . 86
Input Pin Description . . . 89
Structures Used . . . 90
Output Pin Description . . . 91
Part IV Equipment Control Functions 93
Chapter 12 FB_Cyclic_Monitoring: Cyclic Monitoring 95
FB_Cyclic_Monitoring Function Block . . . 96
Input Pin Description . . . 98
Output Pin Description . . . 98
Instantiation and Usage Example . . . 99
Chapter 13 FB_DeadBand: Suppressing Amplitude Oscillations 101
FB_DeadBand Function Block . . . 102
Input Pin Description . . . 104
Output Pin Description . . . 104
Chapter 14 FB_Limiter: Limiting Input Signals 105
FB_Limiter Function Block . . . 106
Input Pin Description . . . 108
Output Pin Description . . . 108

Chapter 15 FB_PWM: Providing a PWM Output 109
FB_PWM Function Block 110
Input Pin Description 116
Structure Used 116
Output Pin Description 117
Chapter 16 FB_Redundant_Sensor_Monitoring: Redundant Sensor Monitoring 119
FB_Redundant_Sensor_Monitoring Function Block 120
Input Pin Description 122
Output Pin Description 125
Chapter 17 FB_Scaling: Scaling Input Signals 127
FB_Scaling Function Block 128
Input Pin Description 130
Output Pin Description 130
Chapter 18 FB_Sensor_Monitoring: Sensor Monitoring 131
FB_Sensor_Monitoring Function Block 132
Input Pin Description 133
Output Pin Description 134
Instantiation and Usage Example 134
Part V Filtering Functions 137
Chapter 19 Global Parameter List 139
Global Parameter List (GPL) 139
Chapter 20 Filter_AnalogInput: Checking Analog Input Varaibility . . 141
Filter_AnalogInput Function Block 142
Input Pin Description 144
Output Pin Description 144
Chapter 21 Filter_Arithmetic: Giving Arithmetic Mean Value 145
Filter_Arithmetic Function Block 146
Input Pin Description 148
Output Pin Description 148
Chapter 22 Filter_MovingAverage: Giving Moving Mean Value 149
Filter_MovingAverage Function Block 150
Input Pin Description 152
Output Pin Description 152

Chapter 23 Filter_PT1: Providing PT1 Transfer Function 153
Filter_PT1 Function Block 154
Input Pin Description 157
Output Pin Description 157
Instantiation and Usage Example 158
Part VI Flip-Flops 163
Chapter 24 JK_FlipFlop: Resetting/Setting Input to Flip Flop Output . 165
JK_FlipFlop Function Block 165
Chapter 25 JK_FlipFlop_MasterSlave: Resetting/Setting Input to Flip Flop Output 167
JK_FlipFlop_MasterSlave Function Block 167
Chapter 26 RS_FlipFlop: Resetting/Setting of Flip Flop Input/Output . 171
RS_FlipFlop Function Block 171
Chapter 27 SR_FlipFlop: Resetting/Setting of Flip Flop Input/Output . 173
SR_FlipFlop Function Block 173
Chapter 28 Toggle_FlipFlop: Toggling of Flip Flop Input/Output 175
Toggle_FlipFlop Function Block 175
Part VII Mathematical Functions 177
Chapter 29 Analysis: Calculating Integral and Derivative Values 179
Analysis Function Block 179
Chapter 30 Frequency_Multiplier: Implementing 32 Blinkers 181
Frequency_Multiplier Function Block 182
Without Hold Condition Description 184
Functionality With Condition Description 185
Chapter 31 Frequency_Output: Implementing a Frequency 187
Frequency_Output Function Block 187
Chapter 32 Normalizer_With_Limiter: Scaling the Minimum and Maximum Input 195
Normalizer_With_Limiter Function Block 195
Chapter 33 ONE_SEC_PULSE: Providing Pulses Every One Second 199
ONE_SEC_PULSE Function Block 199
Chapter 34 Quantizer: Digitalizing the Input Value for the Interval . . . 201
Quantizer Function Block 201
Chapter 35 Signal_Saturation: Limiting to Upper and Lower Saturation Limit 203
Signal_Saturation Function Block 203

Chapter 36 Signal_Statistics: Calculating Maximum, Minimum, Average and Variance 209
Signal_Statistics Function Block 209
Chapter 37 Check_Divisor: Checking for Zero Division Condition . . 213
Check_Divisor Function Block 213
Part VIII Numerical Conversion Functions 215
Chapter 38 ArrayOfByte_TO_String: Converting Array in Byte Format to String Format 217
ArrayOfByte_TO_String Function. 217
Chapter 39 DT_AS_WORD: Converting Date and Time as Word. . . 221
DT_AS_WORD Function Block 221
Chapter 40 DWORD_AS_WORD: Splitting the Double Word into Two Words 223
DWORD_AS_WORD Function Block 223
Chapter 41 String_TO_ArrayOfByte: Output Array and ASCII Value of the Input String 225
String_TO_ArrayOfByte Function. 225
Chapter 42 WORD_AS_DWORD: Shifting Higher Word and Adding Lower Word 229
WORD_AS_DWORD Function Block 229
Part IX Physical Conversion 231
Chapter 43 Celsius_TO_Fahrenheit: Converting Celsius to Fahrenheit. 233
Celsius_TO_Fahrenheit Function. 233
Chapter 44 Celsius_TO_Kelvin: Converting Celsius to Kelvin. 235
Celsius_TO_Kelvin Function Block 235
Chapter 45 Fahrenheit_TO_Celsius: Converting Fahrenheit to Celsius 237
Fahrenheit_TO_Celsius Function. 237
Chapter 46 Frequency_TO_Period: Calculating Period of Time of Given Frequency 239
Frequency_TO_Period Function. 239
Chapter 47 Kelvin_TO_Celsius: Converting Kelvin to Celsius. 241
Kelvin_TO_Celsius Function Block 241
Chapter 48 Period_TO_Frequency: Calculating Frequency of Given Time 243
Period_TO_Frequency Function. 243

Part X Utilities 245
Chapter 49 Hour_Meter: Accumulating Operating Hours 247
Hour_Meter Function Block 248
Input Pin Description 250
Output Pin Description 251
Input – Output Pin 251
Structures Used 252
Control Word Bit Description 252
Status Word 253
Instantiation and Usage Example 254
Chapter 50 Operation_Mode: Selecting Operating Mode 257
Operation_Mode Function Block 258
Input Pin Description 259
Output Pin Description 260
Control Word Bit Description 261
Status Word 262
Part XI Valve Control 263
Chapter 51 Bistable_Valve: Controlling the Bistable Valves 265
Bistable_Valve Function Block 266
Input Pin Description 268
Output Pin Description 269
Structure Used 269
Control Word Bit Description 270
Status Word 271
Instantiation and Usage Example 272
Chapter 52 Monostable_Valve: Controlling Monostable Valves 273
Monostable_Valve Function Block 274
Input Pin Description 276
Output Pin Description 277
Structure Used 277
Control Word Bit Description 278
Status Word 278
Chapter 53 Proportional_Valve: Controlling Proportional Valves 279
Proportional_Valve Function Block 280
Input Pin Description 282
Output Pin Description 283
Input - Output Pin 284

Structures Used 284
Control Word Bit Description 285
Status Word 286
Instantiation and Usage Example 287
Glossary 289
Index 295

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

BEFORE YOU BEGIN

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

⚠ WARNING

UNGUARDED EQUIPMENT

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

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

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

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

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

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

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

START-UP AND TEST

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

⚠ WARNING

EQUIPMENT OPERATION HAZARD

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

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

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

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

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

Before energizing equipment:

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

OPERATION AND ADJUSTMENTS

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

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

About the Book

At a Glance

Document Scope

This document describes the functions of the EcoStruxure Machine Expert Toolbox Library.

Validity Note

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

Product Related Information

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

⚠ WARNING

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.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction

Standard	Description
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2015	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2016	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

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

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

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

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

Part I
Function Requirements

Chapter 1
Function Requirements

Toolbox Function Library Requirements

Hardware Requirements

The Toolbox Library can be used with all Schneider Electric controllers managed with EcoStruxure Machine Expert software.

Using the Library

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Verify the EcoStruxure Machine Expert libraries contained in your program are the correct version after updating EcoStruxure Machine Expert software.
- Verify that the library versions updated are consistent with your application specifications.

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

For more detailed information, see Schneider Electric Libraries *(see EcoStruxure Machine Expert, Libraries Overview)*.

Part II
Bit Functions

Overview

This part describes the bit function family.

What Is in This Part?

This part contains the following chapters:

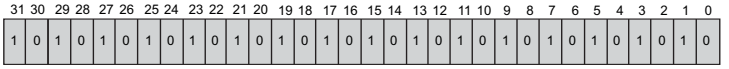
Chapter	Chapter Name	Page
2	Bit Organization	25
3	SetBitTo: Setting/Resetting One Bit	27
4	TestBit: Testing One Bit	29

Chapter 2
Bit Organization

Bit Organization for DWORD

Bit Organization for DWORD

This figure shows the codage rule of the bit position in a DWORD. An example is given for the 16#AAAAAAAA corresponding to the value of 2863311530.



Integer Data Types Description

This table shows the integer data types. Each of the different number types covers a different range types.

Type	Lower limit	Upper limit	Memory space
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
LWORD	0	$2^{64}-1$	64 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32768	32767	16 Bit
UINT	0	65535	16 Bit
DINT	-2147483648	2147483647	32 Bit
UDINT	0	4294967295	32 Bit
LINT	-2^{63}	$2^{63}-1$	64 Bit
ULINT	0	$2^{64}-1$	64 Bit

REAL/LREAL Data Types Description

This table shows the REAL/LREAL data types. REAL and LREAL are called floating-point types. They are required to represent rational numbers.

Type	Range	Resolution	Memory space
REAL uses 4 bytes	-3.402e+38...3.402e+38 (-2^128...2^128)	1,175e-38 (2^-126)	32 Bit
LREAL uses 8 bytes	-1.797e+308...1.797e+308 (-2^1024...2^1024)	2,225e-308 (2^-1022)	64 Bit

NOTE: The support of data type LREAL depends on the target device. Please see in the corresponding documentation whether the 64 bit type LREAL gets converted to REAL during compilation (possibility with a loss of information) or persists.

Chapter 3
SetBitTo: Setting/Resetting One Bit

SetBitTo Function

Pin Diagram

This figure shows the pin diagram of the SetBitTo function:



Functional Description

The SetBitTo function sets/resets (according to set) one bit specified by the bit in the given DWORD input. The bits are counted from low to high starting with 0.

The valid range is 0 to 31.

Input Pin Description

This table describes the input pins of the SetBitTo function:

Input	Data Type	Description
i_dwIput	DWORD	Input value Range: 0...4294967295
i_iPos	INT	Bit position Range: 0...31
i_xSet	BOOL	TRUE: Set FALSE: Reset.

Output Pin Description

This table describes the output pins of the SetBitTo function:

Output	Data Type	Description
SetBitTo	DWORD	Output value Range: 0...4294967295

Limitations

If the i_iPos input is not within the valid range, the input will be interpretated in modulo mode.

Chapter 4
TestBit: Testing One Bit

TestBit Function

Pin Diagram

This figure shows the pin diagram of the TestBit function:



Functional Description

The TestBit function tests one bit specified by the bit in the given DWORD input. The bits are counted from low to high starting with 0.

The output shows the status of presence of bit in that specified position. The valid range is 0 to 31.

Input Pin Description

This table describes the input pins of the TestBit function:

Input	Data Type	Description
i_dwIput	DWORD	Input value Range: 0...4294967295
i_iPos	INT	Bit position Range: 0...31

Output Pin Description

This table describes the output pins of the TestBit function:

Output	Data Type	Description
TestBit	BOOL	Result is True or False

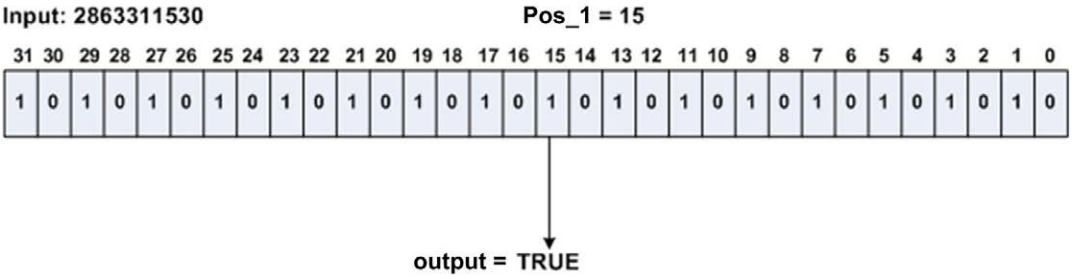
Limitations

If the i_iPos input is not within the valid range, the input will be interpretated in modulo mode.

Usage Example



This figure shows an example of the `TestBit` function:



Part III
Closed Loop Functions

Overview

The part describes the closed loop function family.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
5	FB_2points: Closed Loop 2 Point Transfer Element	33
6	FB_3points: Closed Loop 3 Points Transfer Element	39
7	FB_3points_Ext: Closed Loop Extension For 3 Points Transfer Element	45
8	FB_P: Control Loop With Proportional Only Algorithm	53
9	FB_PI: Proportional and Integration Process Control	59
10	FB_PID: Manual Mode Operation Process Control	67
11	FB_PI_PID: Cascaded PI_PID Control Loop	85

Chapter 5
FB_2points: Closed Loop 2 Point Transfer Element

Overview

This chapter describes the `FB_2points` transfer element function block.

What Is in This Chapter?

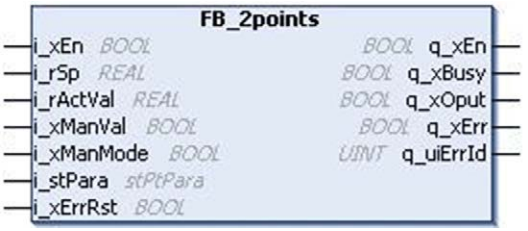
This chapter contains the following topics:

Topic	Page
`FB_2points` Function Block	34
Operating Modes	34
Input Pin Description	37
Structure Used	37
Output Pin Description	38

FB_2points Function Block

Pin Diagram

This figure shows the pin diagram of the FB_2points function block:



Functional Description

The FB_2points function block provides a 2 point transfer function based on hysteresis input.

Operating Modes

Automatic Mode

In automatic mode (i_xEn is TRUE and i_xManMode is FALSE), if the calculated process error is greater than 50% of the hysteresis in the positive direction, then q_xOput is TRUE.

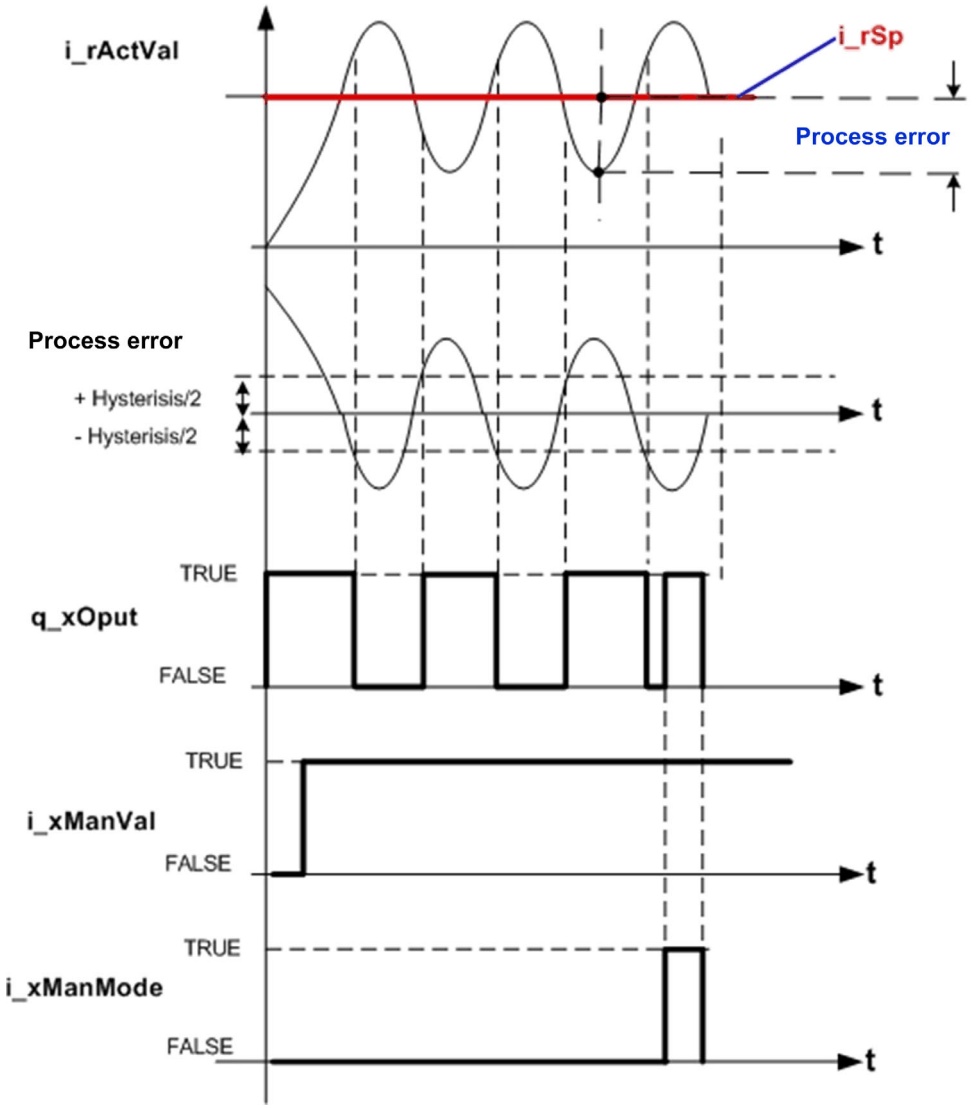
q_xOput is reset only if the calculated process error goes below 50% of hysteresis in negative direction.

Process error = Setpoint value – Actual value.

Manual Mode

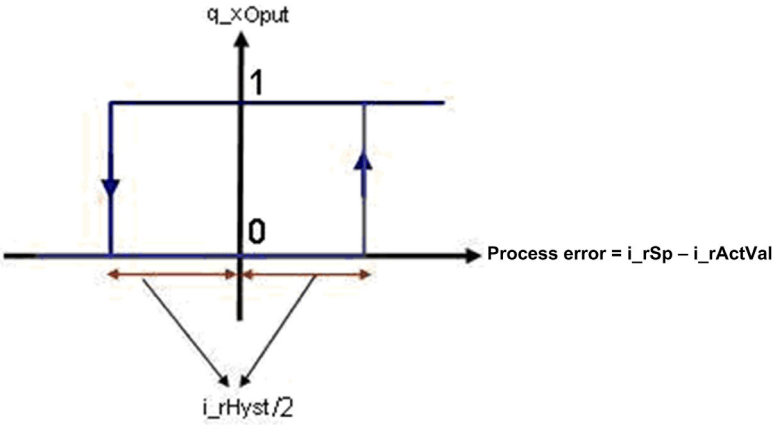
If i_xManMode is TRUE regardless of the current inputs, the output is set to the state of i_xManVal.

This figure shows the timing diagram for the `FB_2points` function block.



Mode Timing Diagram

This figure shows the transfer function for the `FB_2points` function block.



Detected Error State

An invalid parameter at the function block inputs results in a detected error and the corresponding detected error ID is generated.

During the error detected state, the output value is set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Table

This table describes the input pins of the FB_2points function block.

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block FALSE: Disables the function block
i_rSp	REAL	Set point value Range: ±3.4e^{+38}
i_rActVal	REAL	Actual value Range: ±3.4e^{+38}
i_xManVal	BOOL	Manual value (Optional)
i_xManMode	BOOL	Manual mode (Optional)
i_stPara	STRUCT stPtPara	Structure parameter (Please refer the table below).
i_xErrRst	BOOL	Reset for detected error (Rising edge resets detected error.) (Optional)

Structure Used

stPtPara

Structure Element	Type	Description
tCyclTime	TIME	Task cycle time Range: 1...1e^{32} ms
rHyst	REAL	Range of the hysteresis should be greater than 0.0 Range: ±3.4e^{+38}

Output Pin Description

Output Pin Table

This table describes the output pins of the FB_2points function block.

Output	Data Type	Description
q_xEn	BOOL	TRUE: function block enabled FALSE: Disabled
q_xBusy	BOOL	TRUE: function block active and no detected error. FALSE: function block disabled or function block detected error
q_xOput	BOOL	TRUE: If process error is greater than 50% of hysteresis FALSE: If q_xOput is TRUE and detected error goes below 50% of hysteresis in negative direction.
q_xErr	BOOL	TRUE: During detected error FALSE: If no detected error
q_uiErrId	UNIT	0 = No detected error 1 = Invalid parameter task Cycle time = 0 2 = Invalid parameter i_rHyst <= 0.

Chapter 6
FB_3points: Closed Loop 3 Points Transfer Element

Overview

This chapter describes the `FB_3points` transfer element function block.

What Is in This Chapter?

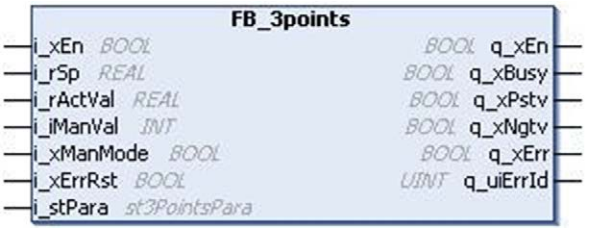
This chapter contains the following topics:

Topic	Page
`FB_3points` Function Block	40
Operating Modes	41
Input Pin Description	43
Structure Used	43
Output Pin Description	44

FB_3points Function Block

Pin Diagram

This figure shows the pin diagram of the FB_3points function block:



Functional Description

The FB_3points function block provides a 3-point transfer element in the functional diagram.

Operating Modes

Automatic mode

This function block checks the value of process error (set point - actual value).

If the process error is positive and more than the upper threshold value for positive side `rPstvOutOn`, it sets the `q_xPstv` output signal.

`q_xPstv` output is reset if the process error decreases below the lower threshold value for positive side `rPstvOutOff`.

If process error is negative, `q_xNgtv` output signal is set upon overshooting upper threshold value for negative side `rNgtvOutOn`.

`q_xNgtv` is reset upon decreasing below lower threshold value for negative side `rNgtvOutOff`.

Manual Mode

The function block output is set manually according to the value of the `i_iManVal` input pin.

If

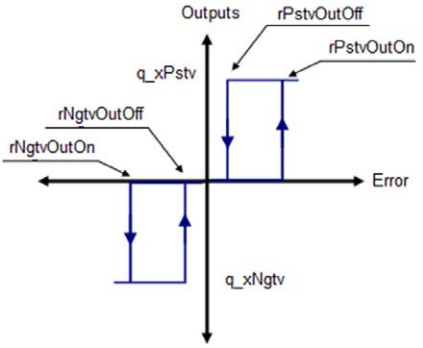
`i_iManVal` >= 1 then `q_xPstv` = TRUE, `q_xNgtv` = FALSE.

`i_iManVal` <= -1 then `q_xPstv` = FALSE, `q_xNgtv` = TRUE.

Else

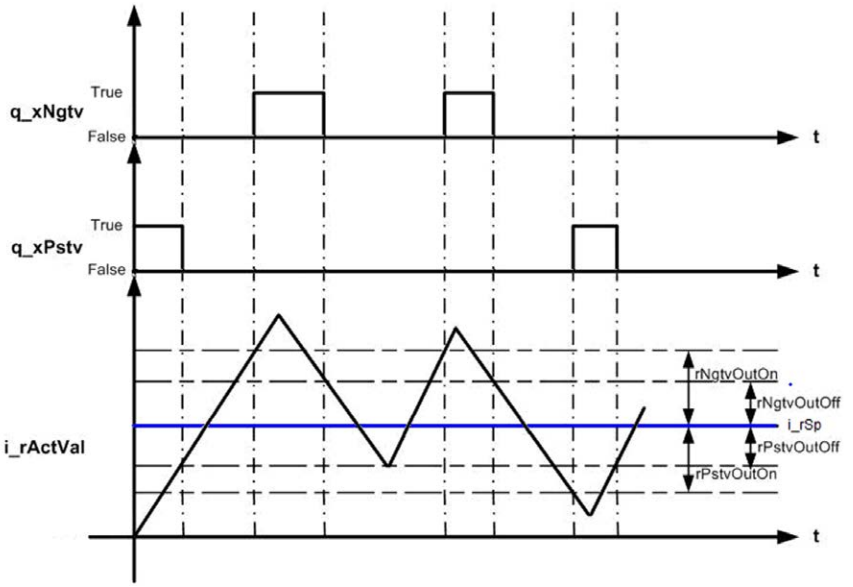
`q_xPstv` = FALSE, `q_xNgtv` = FALSE

This figure shows the transfer function for the `FB_3points` function block:



Timing Diagram

This figure shows the timing diagram for the `FB_3points` function block:



Detected Error State

An invalid parameter at the function block inputs results in a detected error and corresponding detected error ID is generated.

During the error detected state the output values are set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Table

This table describes the input pins of the FB_3points function block:.

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block FALSE: function block disabled
i_rSp	REAL	Set point value of the process Range: ±3.4e^{+38}
i_rActVal	REAL	Actual value of the process Range: ±3.4e^{+38}
i_iManVal	INT	Input value for manual mode. Range: -32768...32767
i_xManMode	BOOL	TRUE: function block in manual mode FALSE: function block in automatic mode (Optional)
i_xErrRst	BOOL	Reset for detected error (Rising edge triggered.) (Optional)
i_stPara	STRUCT st3PointsPara	Function block structure variable

Structure Used

st3PointsPara

Structure Element	Type	Description
rPstvOutOn	REAL	Upper threshold value for positive side of process error Range: 0...1e^{38}
rPstvOutOff	REAL	Lower threshold value for positive side of process error Range: 0...1e^{38}
rNgtvOutOn	REAL	Upper threshold value for negative side of process error Range: 0...1e^{38}
rNgtvOutOff	REAL	Lower threshold value for negative side of process error Range: 0...1e^{38}

Output Pin Description

Output Pin Table

This table describes the output pins of the FB_3points function block:.

Output	Data Type	Description
q_xEn	BOOL	TRUE: Function block enabled FALSE: Disabled
q_xBusy	BOOL	TRUE: Function block active and no detected error. FALSE: Function block disabled or detected error
q_xPstv	BOOL	This output from the 3-points element is TRUE if the upper branch of the hysteresis curve is active.
q_xNgtv	BOOL	This output from the 3-points element is TRUE if the lower branch of the hysteresis curve is active.
q_xErr	BOOL	Detected error signal
q_uiErrId	UINT	Supplies the detected error Id when q_xErr output is set. Range: 0...3

q_uiErrId

This unique integer value indicates some particular detected error:

Detected error ID	Description
0	No detected error
1	Invalid parameter values (If rPstvOutOn < 0 or rPstvOutOff < 0 or rNgtvOutOff < 0 or rNgtvOutOn < 0)
2	Invalid parameter values (rPstvOutOn < rPstvOutOff or rNgtvOutOn < rNgtvOutOff)
3	Internal detected error (function block in unknown state)

Chapter 7
FB_3points_Ext: Closed Loop Extension For 3 Points Transfer Element

Overview

This chapter describes the `FB_3points_Ext` function block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
`FB_3points_Ext` Function Block	46
Operating Modes	47
Input Pin Description	50
Structure Used	50
Output Pin Description	51

FB_3points_Ext Function Block

Pin Diagram

This figure shows the pin diagram of the FB_3points_Ext function block:



Functional Description

The FB_3points_Ext function block provides a 3-point transfer element in the functional diagram.

This function block is an extension for FB_3points function block. It produces a control output q_rOput in the form of 3-point hysteresis loop. The output depends upon the process error and the gain and offset value that are given by the user.

Operating Modes

Automatic Mode

This function block checks the value of process error (difference between set point and actual value). If the process error is positive and greater than the upper threshold value `rOutOn`, it calculates the control output as below,

`q_rOput` = Process error x `rGain` + `rOfst`

If the process error decreases below the lower threshold value `rOutOff`, it resets the control output to zero.

Similarly if process error is negative, and its absolute value is more than upper threshold value `rOutOn`, it calculates the control output as below,

`q_rOput` = Process error x `rGain` - `rOfst`

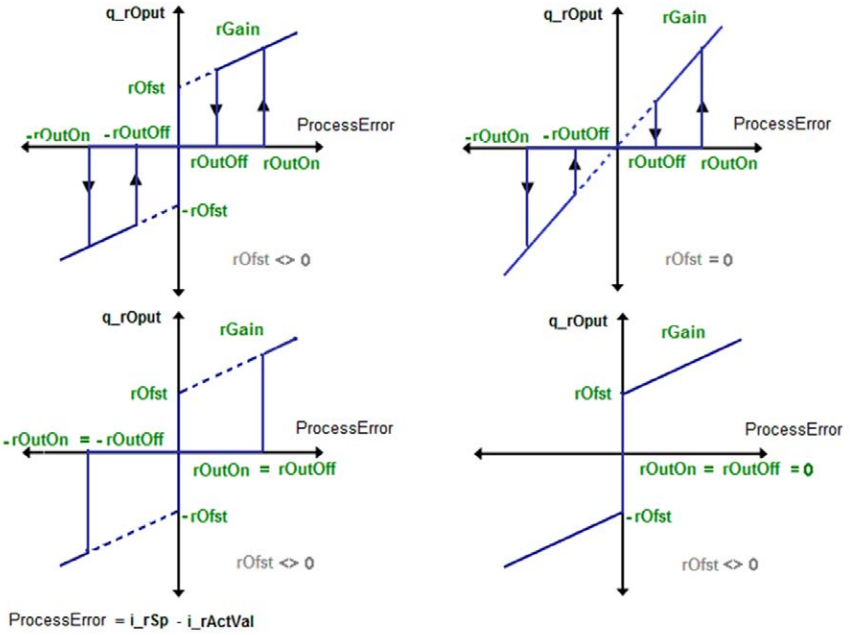
`q_rOput` is reset to zero if absolute value of process error becomes less than lower threshold value `rOutOff`.

Manual Mode

The function block output is set manually as according to the value of input pin `i_rManVa`:.

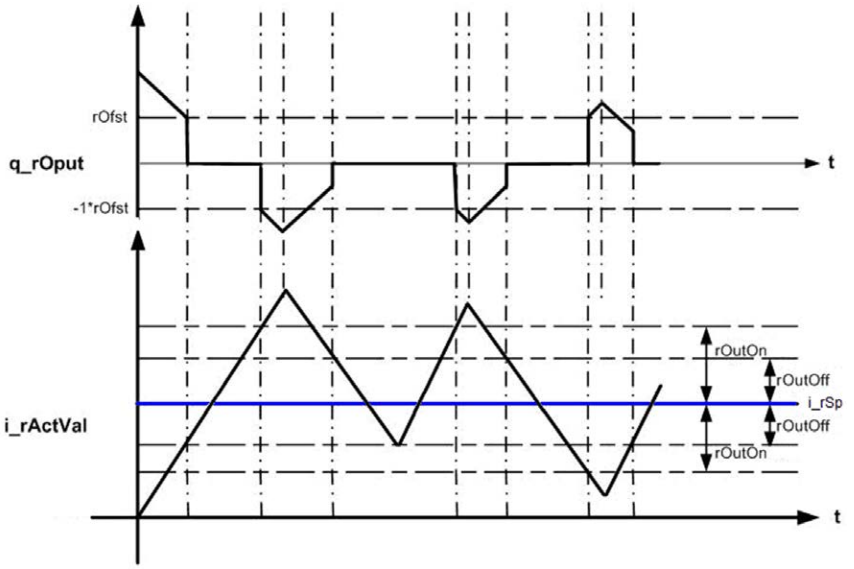
IF	AND IF	THEN
`Abs(i_rManVal)` < 1	-	`q_rOput` = 0.0
`Abs(i_rManVal)` >= 1	`rE` > 0	`q_rOput` = `rE` x `rGain` + `rOfst`
`Abs(i_rManVal)` >= 1	`rE` < 0	`q_rOput` = `rE` x `rGain` - `rOfst`
`Abs(i_rManVal)` >= 1	`rE` = 0	`q_rOput` = 0.0
rE = `i_rSp` - `i_rActVal` **Abs()** Absolute value function.		

This figure shows the transfer function for the `FB_3points_Ext` function block:



Timing Diagram

This figure shows the timing diagram for the `FB_3points_Ext` function block:



Detected Error State

An invalid parameter at the function block input results in a detected error state and a corresponding detected error ID is generated.

During the error detected state the output values are set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Table

This table describes the input pins of the FB_3points_Ext function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block FALSE: function block disabled
i_rSp	REAL	Set point value of the process Range: ±3.4e^{+38}
i_rActVal	REAL	Actual value of the process Range: ±3.4e^{+38}
i_rManVal	REAL	Input value for manual mode. Range: ±3.4e^{+38} (Optional)
i_xManMode	BOOL	TRUE: function block in manual mode FALSE: function block in automatic mode (Optional)
i_xErrRst	BOOL	Reset for detected error (Rising edge triggered) (Optional)
i_stPara	STRUCTst3PtsExtendedPara	Function block structure variable

Structure Used

st3PtsExtendedPara

Structure Element	Type	Description
rOutOn	REAL	Upper threshold value Range: 0...1e^{38}
rOutOff	REAL	Lower threshold value Range: 0...1e^{38}
rGain	REAL	Gain Range: 0...1e^{38}
rOfst	REAL	Offset Range: 0...1e^{38}

Output Pin Description

Output Pin Table

This table describes the output pins of the FB_3points_Ext function block:

Output	DataType	Description
q_xEn	BOOL	TRUE: Function block enabled FALSE: Disabled
q_xBusy	BOOL	TRUE: Function block active and no detected error. FALSE: Function block disabled or detected error
q_rOput	REAL	Output from the function block. Range: $\pm 3.4e^{+38}$
q_xErr	BOOL	Detected error signal

q_uiErrId

This unique integer value indicates some particular detected error:

Detected Error ID	Description
0	No detected error
1	Invalid parameter values (rOutOn < 0 OR rOutOff < 0)
2	Invalid parameter values (rOutOn < rOutOff)
3	Invalid parameter values (rGain < 0)
4	Invalid parameter values (rOfst < 0)
5	Internal detected error (function block in unknown state)

Chapter 8
FB_P: Control Loop With Proportional Only Algorithm

Overview

This chapter describes the FB_P function block.

What Is in This Chapter?

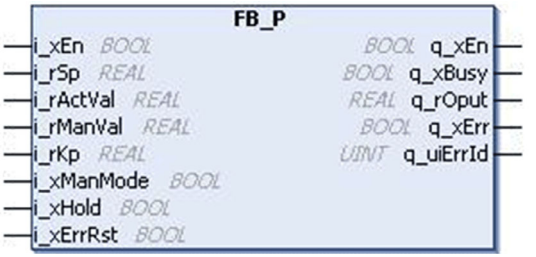
This chapter contains the following topics:

Topic	Page
FB_P Function Block	54
Operation Modes	55
Input Pin Description	57
Output Pin Description	58

FB_P Function Block

Pin Diagram

This figure shows the pin diagram of the FB_P function block:



Functional Description

This FB_P function block is developed to provide a control loop with proportional only algorithm.

Operation Modes

Automatic Mode

This function block provides the proportional response i.e. the output is process error times the gain.

This equation shows the transfer function:

$$G(s) = Kp$$

Where:

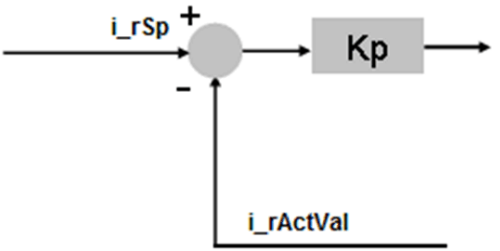
Kp = Proportional gain

q_rOput = G(s) * Process error

Manual Mode

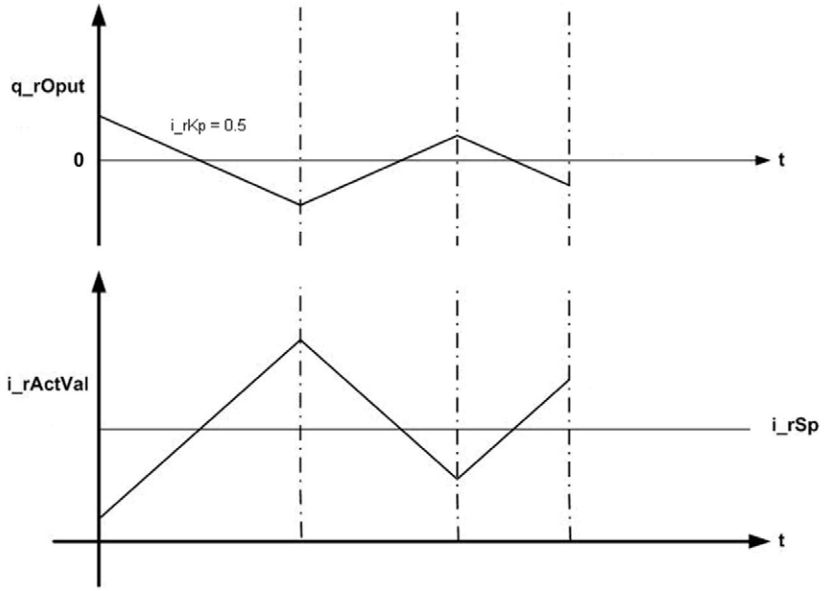
The `q_rOput` function block output is set equal to `i_rManVal`.

This figure shows the function diagram for the `FB_P` function block



Timing Diagram

This figure shows the timing diagram for the FB_P function block



Detected Error State

An invalid parameter at the function block inputs results in detected error and corresponding detected error ID is generated.

During detected error state, the output values are set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE whenever the function block is enabled and there is no detected error.

Input Pin Description

Input Pin Table

This table describes the input pins of the FB_P function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block FALSE: function block disabled
i_rSp	REAL	Set point value of the process Range: ±3.4e^{+38}
i_rActVal	REAL	Actual value of the process Range: ±3.4e^{+38}
i_rManVal	REAL	Input value for manual mode. Range: ±3.4e^{+38} (Optional)
i_rKp	REAL	Proportional gain Range: 0...3.4e^{+38}
i_xManMode	BOOL	TRUE: Function block in manual mode FALSE: Function block in automatic mode (Optional)
i_xHold	BOOL	TRUE: Hold the internal state and output of the function block constant at its current value. FALSE: Disabled (Optional)
i_xErrRst	BOOL	Reset for detected error (rising edge triggered.) (Optional)

Output Pin Description

Output Pin Table

This table describes the output pins of the FB_P function block:

Output	Data Type	Description
q_xEn	BOOL	TRUE: Function block enabled FALSE: Disabled
q_xBusy	BOOL	TRUE: Function block active and no detected error. FALSE: Function block disabled or detected error
q_rOput	REAL	Output from the function block Range: $\pm 3.4e^{+38}$
q_xErr	BOOL	Detected error signal
q_uiErrId	UINT	Supplies the detected error Id when q_xErr output is set. 0: No detected error 1: Invalid parameter values (If i_rKp < 0) 2: Internal detected error (function block in unknown state)

q_uiErrId

This unique integer value indicates some particular detected error

Detected Error ID	Description
0	No detected error
1	Invalid parameter values (i_rKp < 0)
2	Internal detected error (function block in unknown state)

Chapter 9
FB_PI: Proportional and Integration Process Control

Overview

This chapter describes the FB_PI Closed Loop process control function block.

What Is in This Chapter?

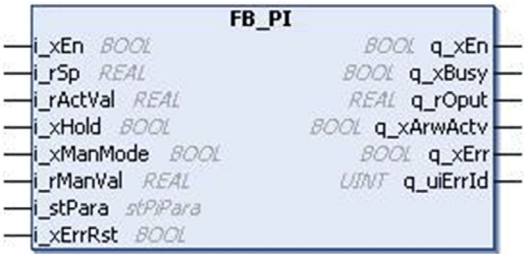
This chapter contains the following topics:

Topic	Page
FB_PI Function Block	60
Input Pin Description	62
Structure Used	64
Output Pin Description	66

FB_PI Function Block

Pin Diagram

This figure shows the pin diagram of the FB_PI function block:



Functional Description

The FB_PI function block is a standard PI function block with manual tuning, hold function, and anti reset wind-up.

The PI controller generates a control output based on the process error in the system (Process error = Set Point – Actual Value). Using the setting of function block parameters, control output can be tuned to reduce the process error.

The proportional and integral value for the process calculated continuously based on the actual value, set point and input parameters. The function block also limits the control output based on limit settings.

Transfer Function

The following equation is the transfer function for the FB_PI function block:

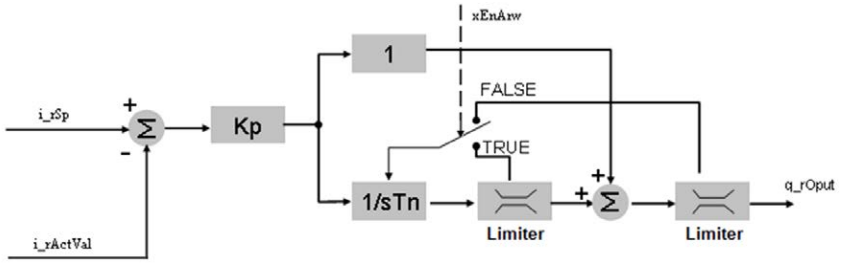
$$G(s) = Kp\left(1 + \frac{1}{sT_n}\right)$$

Where:

K_p = Proportional gain.
sT_n = Integral time

Functional Diagram

This figure shows the functional diagram of the FB_PI function block:



This function block is used to control the Closed Loop processes with continuous input and output variables.

Detected Error State

An invalid parameter at the function block inputs results in a detected error and a corresponding detected error ID is generated.

During the error detected state, the output value is set to zero.

Detected error can be reset only through rising edge of `i_xErrRst` input. The output `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Table

This table describes the input pins of the FB_PI function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block FALSE: Disables the function block
i_rSp	REAL	Set point value Range: ±3.4e^{+38}
i_rActVal	REAL	Actual value Range: ±3.4e^{+38}
i_rManVal	REAL	Manual value Range: ±3.4e^{+38} (Optional)
i_xManMode	BOOL	Manual value (Optional)
i_xHold	BOOL	Hold (Optional)
i_xErrRst	BOOL	Reset for detected error (rising edge reset detected error) (Optional)
i_stPara	STRUCT stPiPara	Structure parameter (Refer to the stPiPara description *(see page 64)*.)

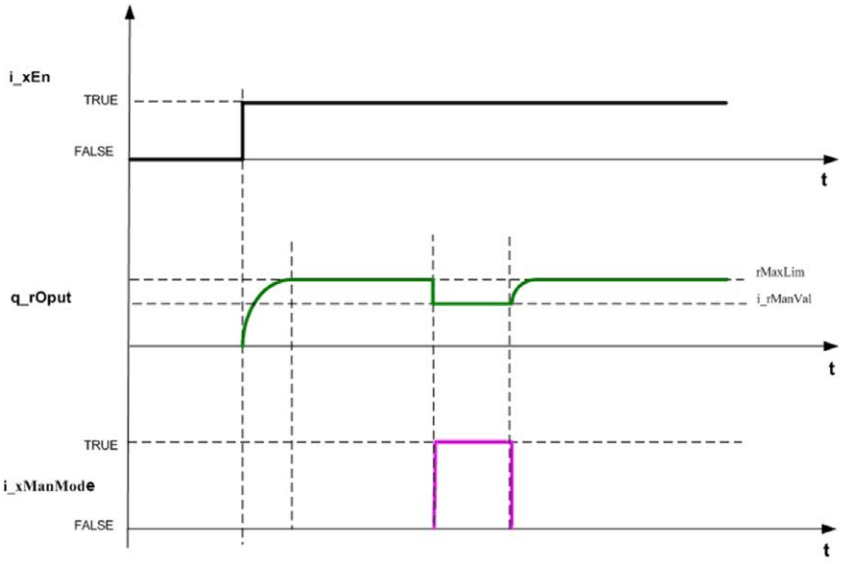
i_xManMode

i_xManMode decides the manual mode of the FB_PI function block.

If the function block is enabled and manual is set to TRUE, then the function block will set the manual value (i_rManVal) as the PI output and stops the PI algorithm as shown in block diagram function block in manual mode.

If the auto mode is enabled, then PI Algorithm executes continuously.

This figure shows the time diagram of the function block in manual mode:



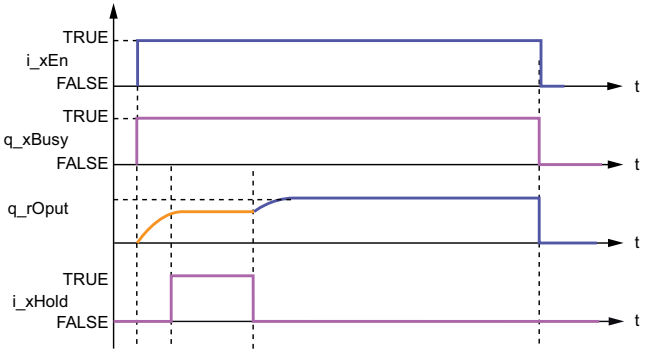
i_xHold

i_xHold will hold the PI output at current level.

If this input is TRUE, then the PI output will be maintained at last value and internal calculations of PI algorithm will be stopped as shown in block diagram function block on hold.

If this input is FALSE, then PI algorithm is executed cyclically. The new PI output will be calculated from the last value.

Time diagram of the function block on hold:



Structure Used

stPiPara

Structure Element	Type	Description
tCyclTime	TIME	Task cycle time Range: 10 ms...60 s
xEnArw	BOOL	Enable anti reset wind-up
tTn	TIME	Integral action time Range: 1...1e^{32} ms
rKp	REAL	Proportional gain Value Range: ±3.4e^{+38}
rMaxLim	REAL	Maximum Output Limit Range: ±3.4e^{+38}
rMinLim	REAL	Minimum Output Limit Range: ±3.4e^{+38}

tCyclTime

tCyclTime is the time between the two executions of the function block. If the task is assigned as cyclic, then it is equal to the task cycle time of the cyclic task.

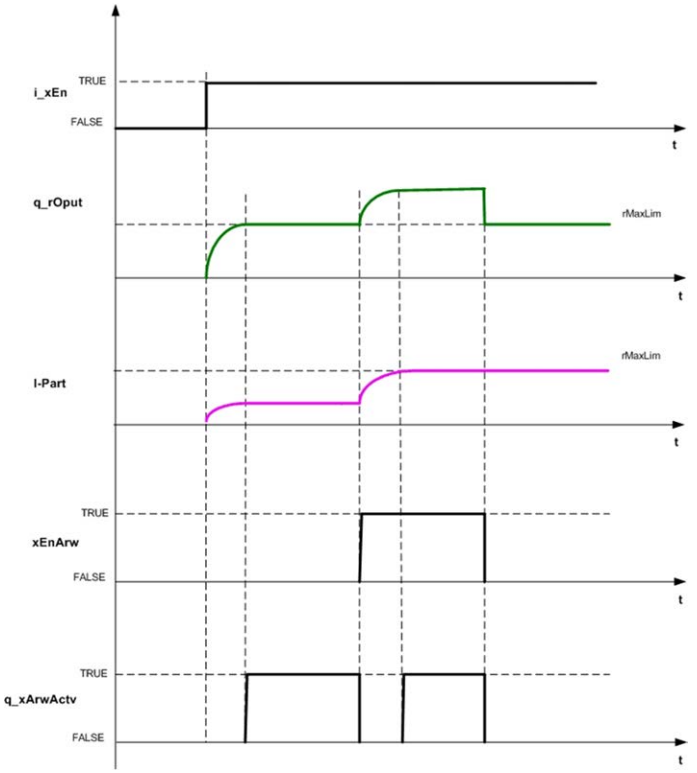
xEnArw

xEnArw will enable anti reset wind-up (ARW) operation.

If FALSE, hold the integral part if the entire control output reaches a limit.

If TRUE, the function block holds the integral part only if the integral part reaches a limit. The output is equal to the sum of limit value and proportional part if integral part reaches a limit as shown in block diagram function block on enable ARW.

This figure shows the function block on Enable ARW mode:



tTn

Integral time for PI loop

rKp

Proportional gain for PI loop

rMaxLim

Output greater than this limit is limited to `rMaxLim` value.

rMinLim

Output less than this limit is limited to `rMinLim` value.

NOTE: If the `rMinLim` is greater than 0 then PI then operation starts from the `rMinLim` value.

Output Pin Description

Output Pin Table

This table describes the output pins of the FB_PI function block.

Output	Data Type	Description
q_xEn	BOOL	TRUE: Function block enabled
q_xBusy	BOOL	TRUE: function block active and no detected error. FALSE: function block disabled or function block detected error
q_rOput	REAL	Calculated PI output Range: $\pm 3.4e^{+38}$
q_xArwActv	BOOL	TRUE: Output is limited, anti reset wind-up is active.
q_xErr	BOOL	Detected error signal
q_uiErrId	UNIT	Detected error ID of particular detected error Range: 0...5

q_uiErrId

This unique integer value indicates the particular detected error.

Detected error ID	Description
0	No detected error
1	Invalid task cycle time = zero
2	Invalid parameter i_rOputMaxLim < i_rOputMinLim
3	Invalid parameter rKp < zero
4	Invalid parameter tTn = zero
5	Internal detected error (function block in unknown state)

Chapter 10
FB_PID: Manual Mode Operation Process Control

Overview

This chapter describes the FB_PID function block.

What Is in This Chapter?

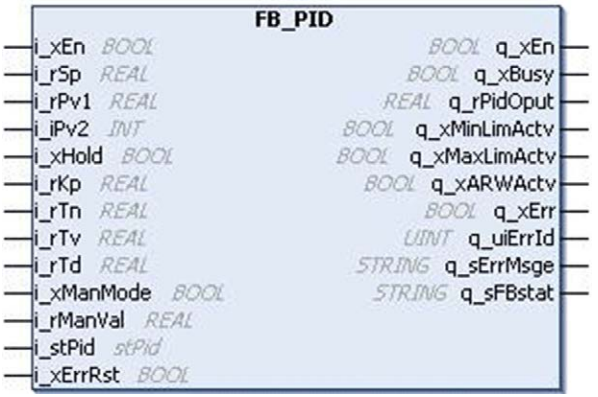
This chapter contains the following topics:

Topic	Page
FB_PID Function Block	68
Input Pin Description	72
Structure Used	74
Output Pin Description	76
Instantiation and Usage Example	78

FB_PID Function Block

Pin Diagram

This figure shows the pin diagram of the FB_PID function block:



Functional Description

The FB_PID function block is a standard PID function block with manual tuning, hold function, bumpless transfer and damping time for derivative action.

This function block provides following features:

- Different modes such as P, PI, PD, and PID.
- Manual mode operation to control the PID output in manual mode.
- Anti reset wind-up to avoid the saturation or wind-up in integral action: If the control variable reaches actuator limit, process error will continue to integrate very large integral term is called as windup.
- Damping time (Td) to filter the overshoot due to derivative action.
- Bump less transfer is activated when mode changes from manual to auto.
  Bump less transfer avoids sudden change in PID output when mode changes.
- Detected error status is generated by function block to display detected errors.
- Inner and outer window functions are used in integral calculations.
  If absolute value of process error is less than inner window then integral part is scaled with a factor [ABS (err)/Inner Window].
  This minimizes the overshoot in the PID output.
  If absolute value of process error is greater than inner window and less than outer window then normal integral calculations are done.
  If absolute value of process error is greater than outer window then anti reset windup is active and integral output will hold the last value.

PID Output

The following equation shows the PID output:

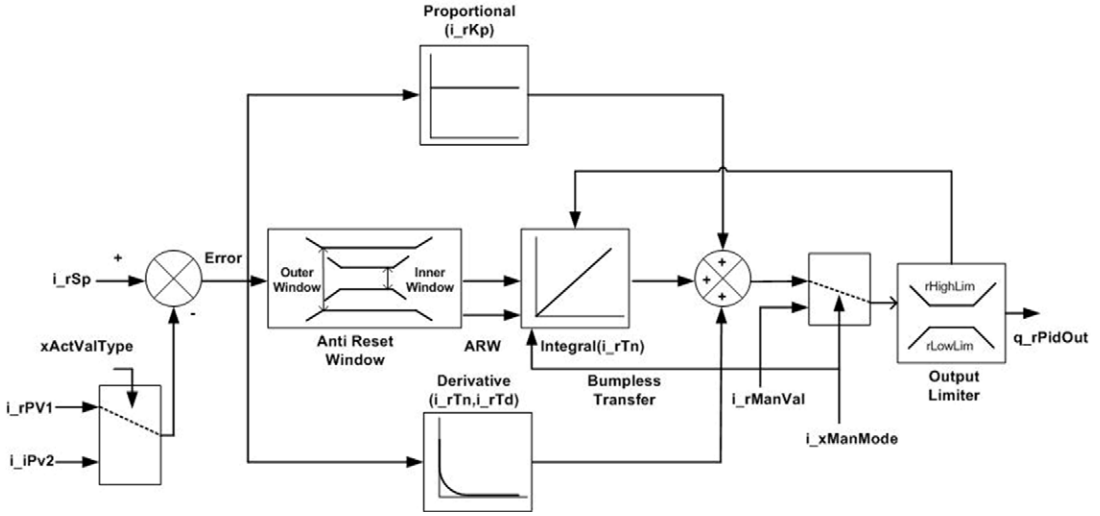
$$y(t) = Kp\left(e(t) + \frac{1}{Tn}\int e(t)dt + \frac{Tv}{1+Td}\frac{de(t)}{dt} \right)$$

Where:

y (t) = PID Output
Kp = Proportional gain
Tn = Integral time
Tv = Derivative time
Td = Filter time for derivative
e (t) = Process error between set point and feedback value.

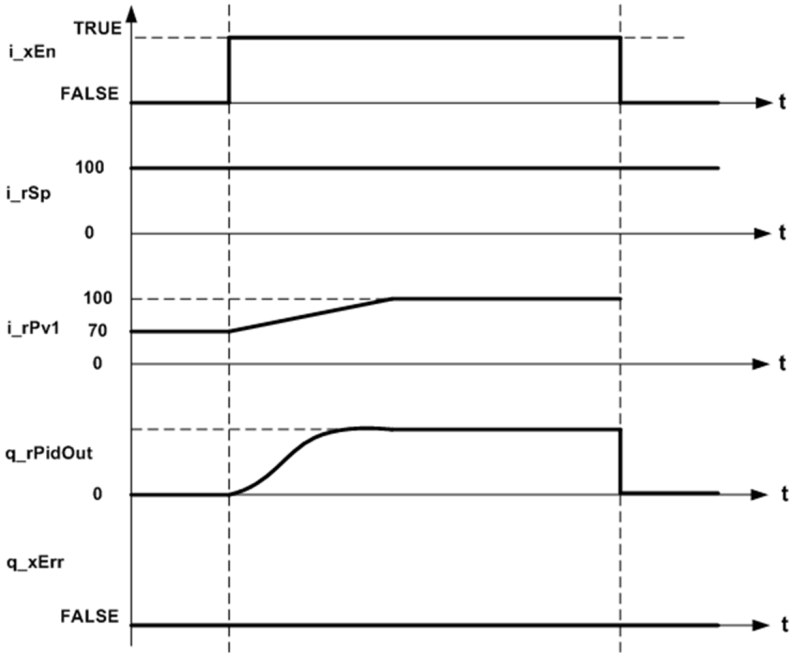
Schematic Diagram

This figure shows the block diagram for the `FB_PID` function block:



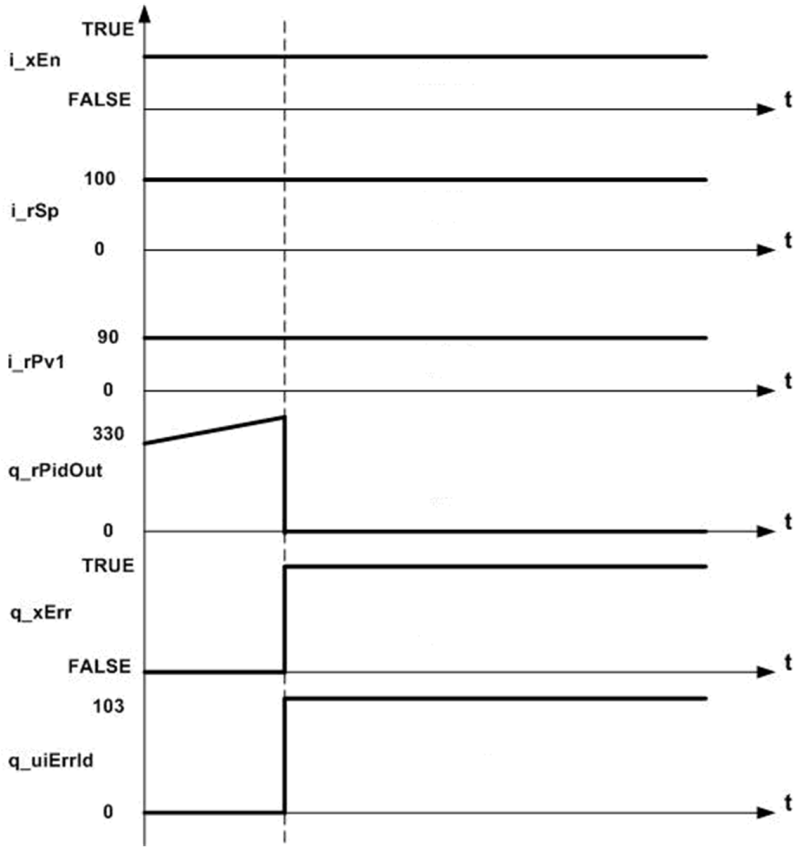
Normal Behavior Diagram

This figure shows normal behavior diagram of the FB_PID function block:



Detected Error Diagram

This figure shows the FB_PID function block diagram with detected error:



Input Pin Description

Input Pin Description

This table describes the input pins of the FB_PID function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables FB_PID function block FALSE: Disables FB_PID function block
i_rSp	REAL	Set point Range: ±3.4e^{+38}
i_rPv1	REAL	Feedback value from the process. Range: ±3.4e^{+38}
i_iPv2	INT	Feedback value from the process. Range: -32768...32767
i_xHold	BOOL	TRUE: Holds the PID output at current level and stops the PID calculations. FALSE: Normal running of PID Mode. (Optional)
i_rKp	REAL	Proportional gain for PID Range: 0.0...3.4e^{+38}
i_rTn	REAL	Integral time for PID Range: 0.0...60000 milliseconds
i_rTv	REAL	Derivative time for PID control Range: 0.0...60000 milliseconds
i_rTd	REAL	Damping time for derivative action Range: 60000.0 > i_rTd > i_stPid.rTargCyclTime
i_xManMode	BOOL	TRUE: Manual mode FALSE: Auto mode (factory setting) (Optional)
i_rManVal	REAL	Manual PID output value when i_xManMode is TRUE. Range: ±3.4e^{+38} (Optional)
i_stPid	STRUCT st_Pid	Parameter structure (Refer to the *(see page 74)*i_stPid description.)
i_xErrRst	BOOL	TRUE: Reset detected error (Optional) (Rising edge resets the detected error)

i_xEn

If TRUE, enables the function block.

If FALSE, then PID output set to zero and detected error status (`q_xErr`) is cleared and detected error id (`q_uiErrId`) is set to zero.

i_xHold

If TRUE, then PID output is maintained at last value and internal calculations of PID algorithm are stopped.

If FALSE, then PID algorithm is executed cyclically.

New PID output is calculated from last value.

i_xErrRst

This resets the current detected error.

On rising edge of this input, detected error is reset.

The PID output is set to zero.

i_xManMode

If TRUE, then PID output is equal to `i_rManVal`.

If FALSE then as per bump less transfer, PID output starts from `i_rManVal` and PID algorithm starts execution.

Structure Used

stPid

Structure Element	Type	Description
xActValType	BOOL	TRUE: i_rPv1 is process value. FALSE: i_iPv2 is process value.
rDbnd	REAL	Dead band value for internal process error calculation. Range: 0.0...100.0 (Optional)
rTargCyclTime	REAL	Target cycle time Range: 60000.0 > rTargCyclTime > 0.0 Unit: ms
rLowLim	REAL	Lower limit of PID output Range: ±3.4e^{+38}
rHighLim	REAL	High limit of PID output Range: ±3.4e^{+38}
rInerWndo	REAL	Inner window for reduced Integral part. Range: 0.0...3.4e^{+38}
rOterWndo	REAL	Outer window for disabling Integral part. Range: 0.0...3.4e^{+38}

NOTE: xActValType, rLowLim, rHighLim, rInerWndo, rOterWndo accept the new state or value change on rising edge of i_xEn.

xActValType

Selects the process value.

rDbnd

Dead band value is used for process error calculation.

If absolute value of process error is greater than dead band then process error value is calculated as: Process error = Set point - Actual Feedback Value.

If absolute value of process error is less than dead band then process error = 0.0.

rTargCyclTime

Target cycle time for PID.

Cycle time can be measured and entered as fix value or current cycle time can be measured in each controller scan as target cycle time.

Parameter `stPid.rTargCyclTime` = 0 results detected error with ID20: Invalid CycleTime

Parameter `stPid.rTargCyclTime` greater than derivative dumping time (Td) results detected error with ID201: Incorrect Td parameter

rLowLim

Lower limit of PID output.

If internal PID output is less than `rLowLim` then PID, output is clamped to `rLowLim`.

rHighLim

Higher limit of PID output.

If internal PID output is greater than `rHighLim` then PID output is clamped to `rHighLim`.

rInerWndo

If absolute value of process error is less than `rInerWndo` then Integral calculations are scaled by factor [ABS (detected error) / Inner Window].

Process error is difference between set point and actual process value.

If process error is greater than `rInerWndo` and less than `rOterWndo` then normal integral calculations are done.

This window is use to reduce overshoot in PID output.

Process error = Set Point - Process value

rOterWndo

If absolute value of process error is greater than `rOterWndo` then Integral calculations are stop and integral output is held at last value.

In the PID output maximum contribution is due to integral action.

If process error is not reducing even though PID output is changing, then PID output saturate due to windup in integral action.

To avoid saturation and windup, `rOterWndo` parameter is required.

Output Pin Description

Output Pin Table

This table describes the output pins of the FB_PID function block:

Output	Data Type	Description
q_xEn	BOOL	TRUE: function block enabled. FALSE: function block disabled.
q_xBusy	BOOL	TRUE: PID is active and no internal detected error. FALSE: PID is not active or detected error.
q_rPidOput	REAL	The output of PID controller. Range: i_stPid.rLowLim...i_stPid.rHighLim
q_xMinLimActv	BOOL	TRUE: If PID output Less than or equal to i_stPid.rLowLim (Minimum limit). FALSE: If PID output greater than i_stPid.rLowLim (Minimum limit).
q_xMaxLimActv	BOOL	TRUE: If PID output greater than or equal to i_stPid.rHighLim (Maximum limit). FALSE: If PID output less than i_stPid.rHighLim (Maximum limit).
q_xARWActv	BOOL	TRUE: Anti reset windup is active. FALSE: Anti reset windup is not active.
q_xErr	BOOL	TRUE: function block detected error FALSE: No detected error.
q_uiErrId	UNIT	Indicates the detected error number when the detected error output is set. Range: 0, 100, 103, 104, 105, 106, 107
q_sErrMsge	STRING	Detected error message
q_sFBstat	STRING	Function block status

q_xEn

True to indicate the enable status of function block.

q_xBusy

TRUE if function block is started without any detected error.

If any detected error occurs, then busy output goes low.

q_rPidOput

The FB_PID starts calculating the output each cycle, if there is no detected error.

q_xARWActv

Indicate status of anti reset windup.

If integral time is greater than zero, integral action is active.

If TRUE, then integral action is stop and integral output is held at last value.

TRUE:

- Case 1: Integral time > 0.0 and ((PID Output >=Maximum limit) and (Process error > 0.0))
- Case 2: Integral time > 0.0 and ((PID Output <=Minimum limit) and (Process error < 0.0))

q_xErr

TRUE indicates the detected error and PID output is set to zero.

q_uiErrId & q_sErrMsge

This gives detected error number and detected error message when `q_xErr` is TRUE.

Detected Error ID	Description
0	No detected error
1	Internal detected error
20	Invalid cycle time
114	Invalid limit parameter
115	Invalid dead band limit
200	Incorrect PID parameter
201	Incorrect Td parameter
202	Incorrect I window

q_sFBstat

FB Active: Function block is active and working without detected error.

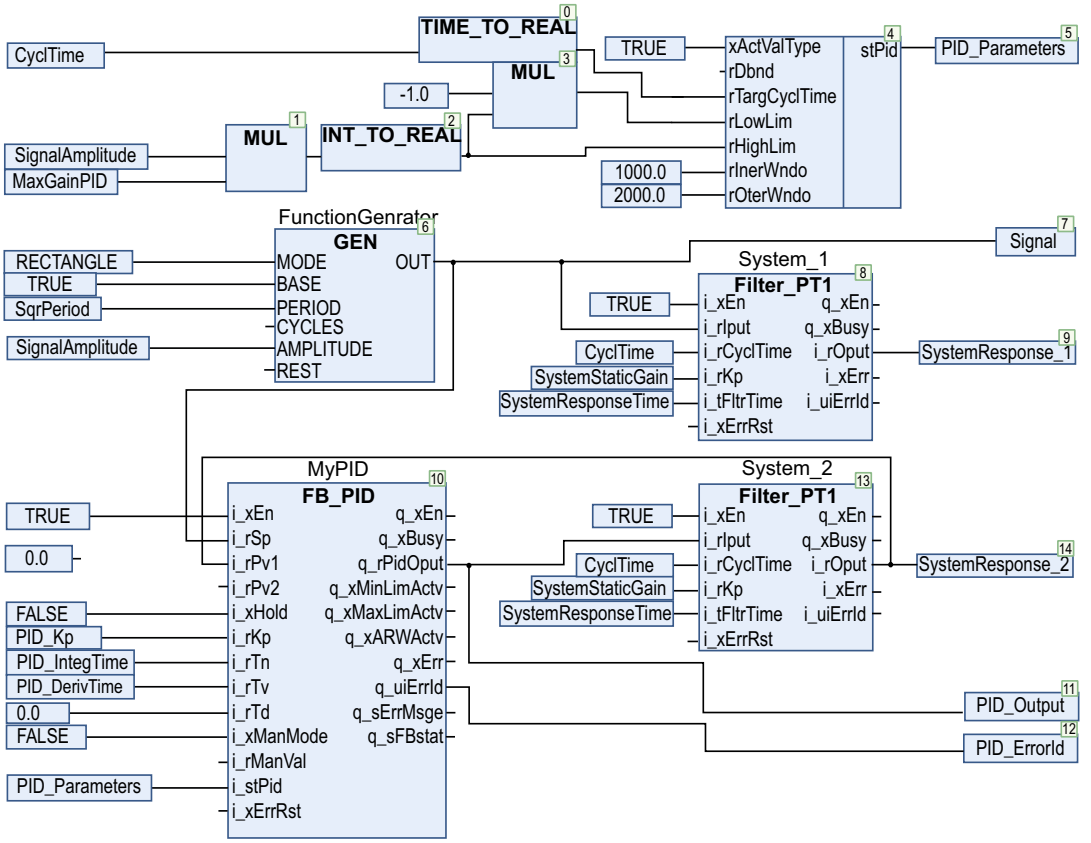
FB Detected error: Function block is active and there is a detected error.

FB Disabled: Function block is disabled.

Instantiation and Usage Example

Instantiation and Usage Example

This figure shows an instance of the FB_PID function block:



- A square signal is generated using the GEN, key parameters are `SqrPeriod` and `SignalAmplitude`.
- The system to control is a simple first order filter, key parameters are `SystemResponseTime` and `SystemStaticGain`.
- A trace is done in open loop `SystemResponse_1` and in closed loop using `FB_PID` function block.

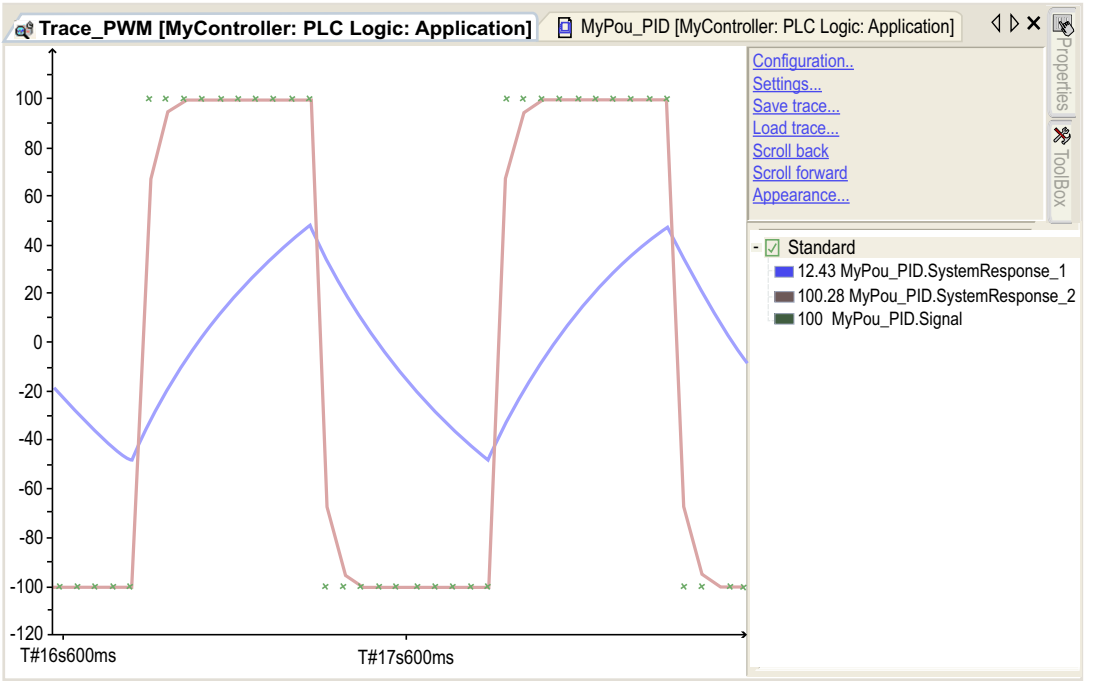
Data of this example are:

Trace_PWM [MyController: PLC Logic: Application] | **MyPou_PID [MyController: PLC Logic: Application]**

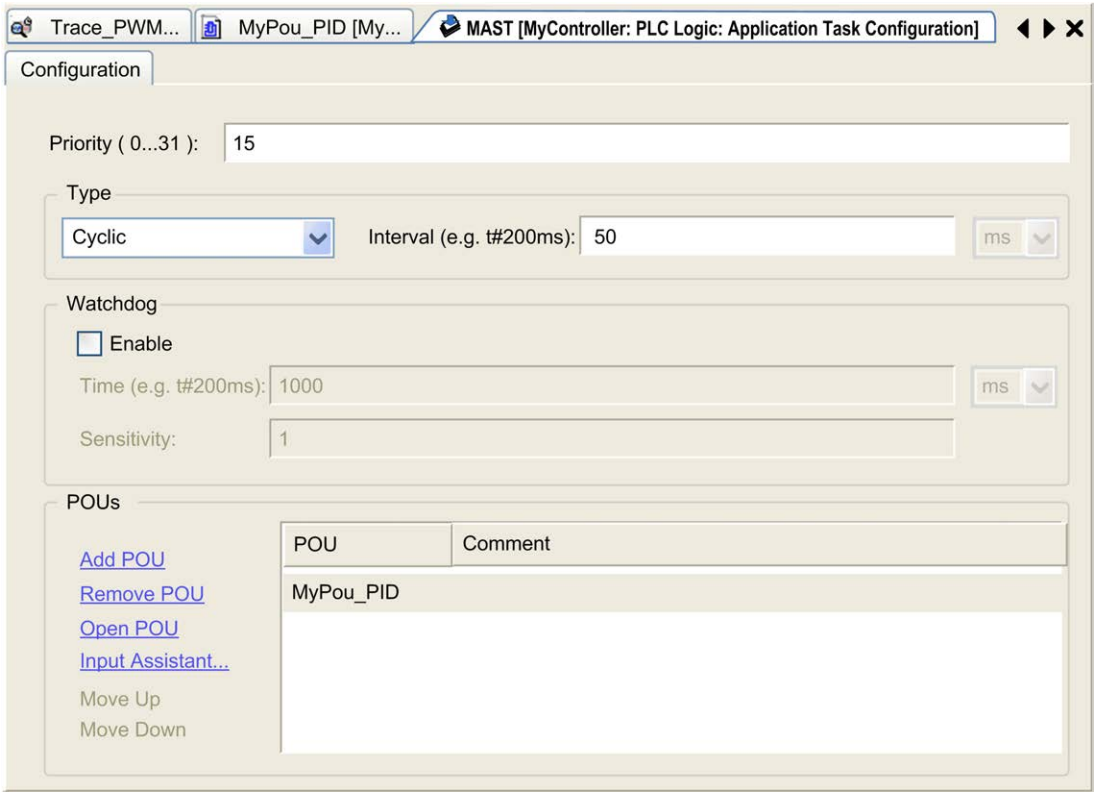
```
PROGRAM MyPou_PID
VAR
 FlagDemarrer        : BOOL;
 FunctionGenerator   : GEN;
 SqrPeriod           : TIME := T#1000MS;
 SignalAmplitude     : INT := 100;
 Signal              : INT;
 System_1            : Filter_PT1;
 System_2            : Filter_PT1;
 CyclTime            : TIME := T#50MS;(* Should have the same value than the periodicity of the POU in the MAST*)
 SystemStaticGain    : REAL := 1.0;
 SystemResponseTime  : TIME := T#500MS;
 SystemResponse_1    : REAL;
 SystemResponse_2    : REAL;

 MyPID               : FB_PID;
 PID_Parameters      : stPid;
 PID_Kp              : REAL := 7.5;
 PID_IntegTime       : REAL := 44.0;
 PID_DerivTime       : REAL := 0.0;
 PID_ErrorId         : UINT;
 PID_Output          : REAL;
 MaxGainPID          : INT := 20;
END_VAR
```

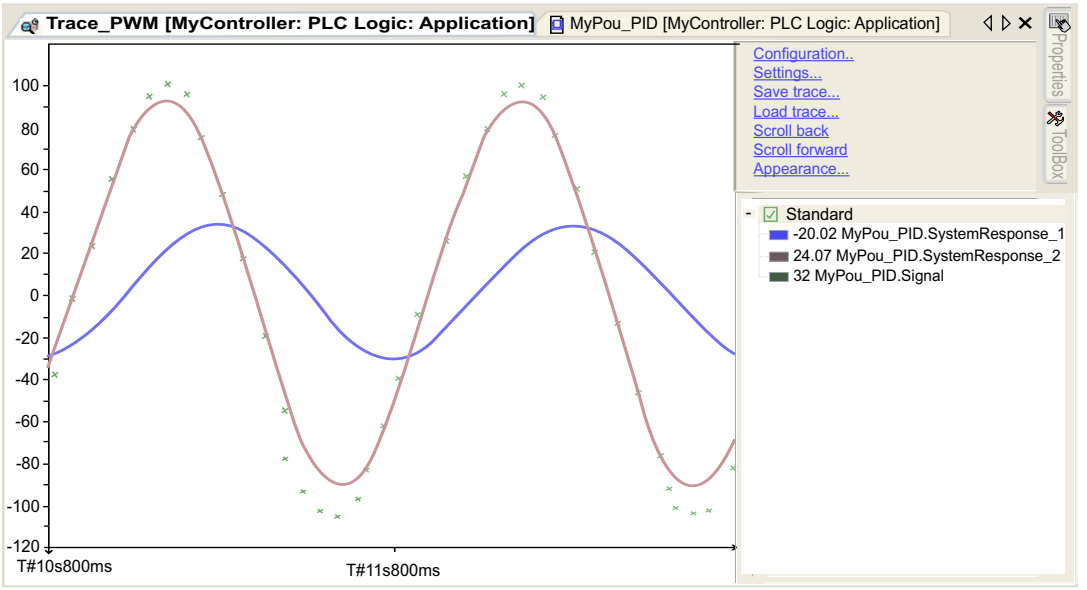
Using the previous setting, the setpoint/open loop/closed loop answer is:



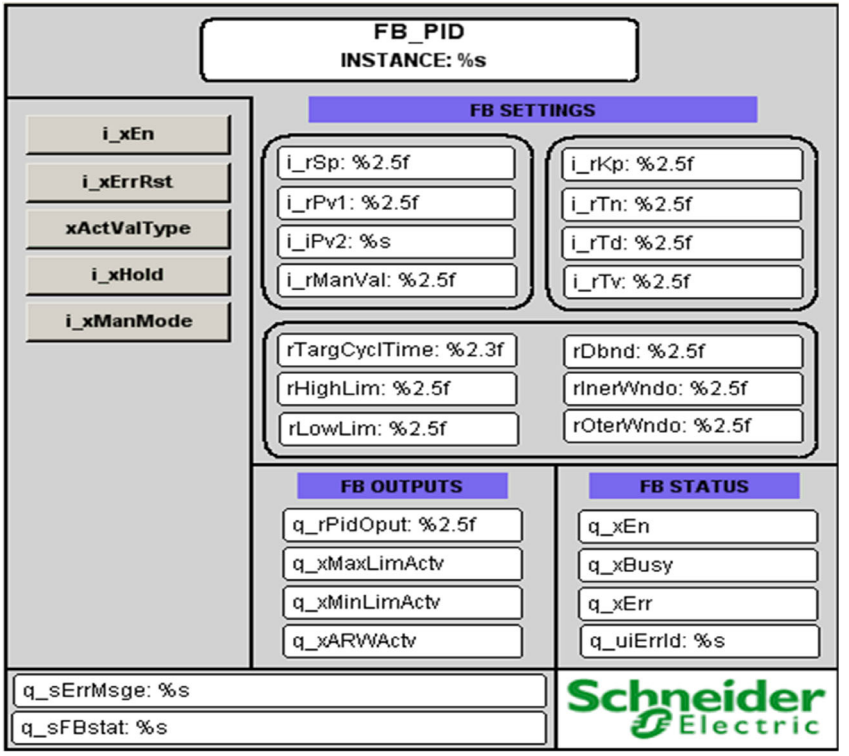
The input `i_tCyclTime` of the first order filters System_1 and System_2 (dataCyclTime) must have exactly the same value as the period of the POU in the MAST, here 50 milliseconds.



When the GEN MODE is changed from RECTANGLE to SINUS with the same other parameters, the Sinus answer is:



This figure shows the visualization of the FB_PID function block:



Detected Error State

This table describes some general detected errors:

Issue	Cause	Solution
Detected error state	Invalid input parameter	Enter valid parameter, then reset detected error

NOTE: If the function block is disabled, outputs are set to zero.

Chapter 11
FB_PI_PID: Cascaded PI_PID Control Loop

Overview

This chapter describes the FB_PI_PID function block.

What Is in This Chapter?

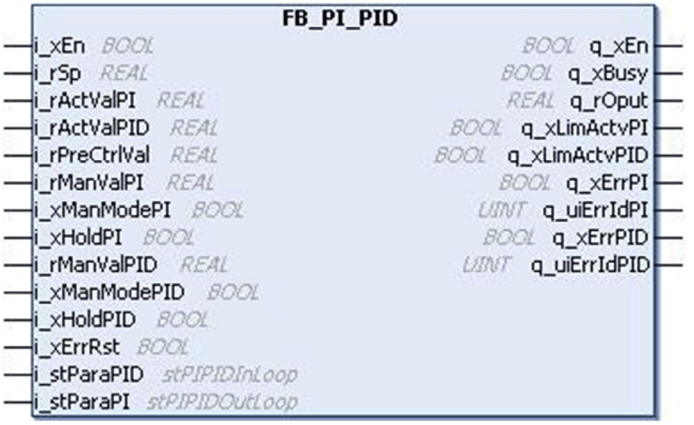
This chapter contains the following topics:

Topic	Page
FB_PI_PID Function Block	86
Operation Modes	86
Input Pin Description	89
Structures Used	90
Output Pin Description	91

FB_PI_PID Function Block

Pin Diagram

This figure shows the pin diagram of the FB_PI_PID function block:



Functional Description

The FB_PI_PID function block provides a cascaded operation of FB_PI, FB_Limiter and FB_PID.

This function block consists of a PI, a control limiter, and a PID element.

Operation Modes

Automatic Mode

The function block calculates the PI response for set point i_rSp and outer loop actual value i_rActValPI. This PI response added to pre control value i_rPreCtrlVal and limited by maximum and minimum threshold inputs, serves as set point to inner PID loop.

The inner loop calculates a PID response with i_rActValPID as the inner loop actual value.

This equation shows the transfer function of PI element:

$$G(s) = Kp(1 + \frac{1}{sT_n})$$

Where:

K_p = Proportional gain
T_n = Integral time

This equation shows the transfer function of PID element:

$$G(s) = Kp(1 + \frac{1}{sT_n} + \frac{sT_v}{1 + sT_d})$$

Where

K_p = Proportional gain
T_n = Integral time

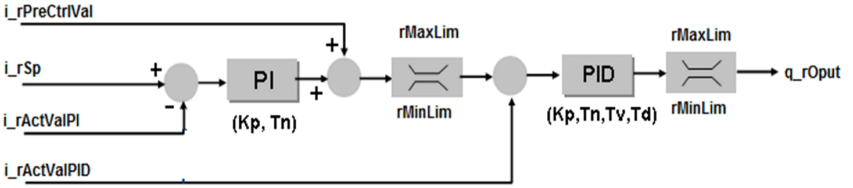
Automatic mode:

T_d = Derivative time
T_v = Filter time

Manual Mode

The PI loop and the PID loop can be setup individually in manual modes using input pins `i_xManModePI` and `i_xManModePID` respectively. In manual mode, PI loop output and PID loop outputs are substituted by values at input pins `i_rManValPI` and `i_rManValPID` respectively.

This figure shows the transfer function for `FB_PI_PID` function block



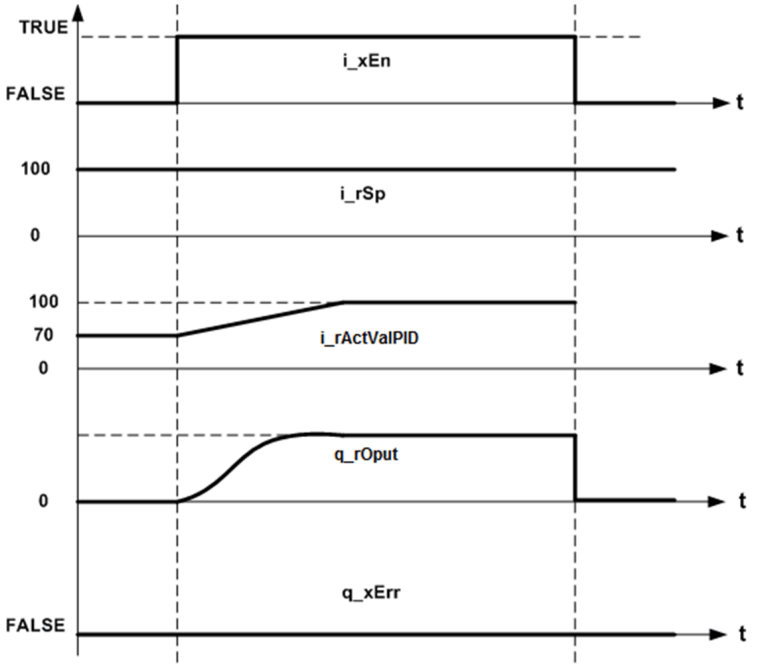
Function block EquipmentControl_limiter

PI Function block ClosedLoop_PI

PID Function block ClosedLoop_PID

Timing Diagram

This figure shows the timing diagram for the FB_PI_PID function block



Detected Error State

An invalid parameter at the function block inputs results in a detected error and a corresponding detected error ID is generated.

During the error detected state the output values are set to zero.

Detected error can be reset only through rising edge of i_xErrRst input.

The output q_xBusy is TRUE whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Table

This table describes the input pins of the FB_PI_PID function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block FALSE: Disables the function block
i_rSp	REAL	Set point value of the process Range: $\pm 3.4e^{+38}$
i_rActValPI	REAL	Actual value of the process to PI outer loop Range: $\pm 3.4e^{+38}$
i_rActValPID	REAL	Actual value of the process to PID inner loop Range: $\pm 3.4e^{+38}$
i_rPreCtrlVal	REAL	Pre control value added to the output from PI outer loop Range: $\pm 3.4e^{+38}$
i_rManValPI	REAL	Manual input for PI outer loop Range: $\pm 3.4e^{+38}$ (Optional)
i_xManModePI	BOOL	TRUE: Operate PI outer loop in manual mode. FALSE: Operate PI outer loop in auto mode (Optional)
i_xHoldPI	BOOL	TRUE: Hold the PI outer loop output and internal state constant FALSE: Disabled (Optional)
i_rManValPID	REAL	Manual input for PID inner loop Range: $\pm 3.4e^{+38}$ (Optional)
i_xManModePID	BOOL	TRUE: Operate PID inner loop in manual mode. FALSE: Operate PID inner loop in auto mode (Optional)
i_xHoldPID	BOOL	TRUE: Hold the PID inner loop output and internal state constant FALSE: Disabled (Optional)
i_xErrRst	BOOL	Reset for detected error (Rising edge resets detected error.) (Optional)
i_stParaPID	STRUCT stPIPIDOutLoop	Control parameters for PID inner loop
i_stParaPI	STRUCT stPIPIDInLoop	Reset for detected error

Structures Used

stPIPIDOutLoop

Structure Element	Type	Description
rKp	REAL	Proportional gain Range: 0.0...1e^{38}
tTn	TIME	Integral time Range: 0...60000 ms
tTv	TIME	Derivative time Range: 0...60000 ms
tTd	TIME	Filter time Range: 0...60000 ms
rMaxLim	REAL	Maximum output limit for PID inner loop Range: ±3.4e^{+38}
rMinLim	REAL	Maximum output limit for PID inner loop Range: ±3.4e^{+38}
rInerWndo	REAL	Inner window for reduced I-part. Range: ±3.4e^{+38}
rOterWndo	REAL	Outer window for disabling I-part. Range: ±3.4e^{+38}

stPIPIDInLoop

Structure Element	Type	Description
tCyclTime	TIME	Task cycle time. Range: 1...60000 ms
rKp	REAL	Proportional gain Range: 0...1e^{38}
tTn	TIME	Integral time Range: 0...60000 ms
rMaxLim	REAL	Maximum output limit for PI outer loop Range: ±3.4e^{+38}
rMinLim	REAL	Minimum output limit for PI outer loop Range: ±3.4e^{+38}

Output Pin Description

Output Pin Table

This table describes the output pins of the FB_PI_PID function block.

Output	Data Type	Description
q_xEn	BOOL	TRUE: Function block is enabled FALSE: Disabled
q_xBusy	BOOL	TRUE: Function block is active and no error is detected. FALSE: function block disabled or detected error
q_rOput	REAL	Output from the cascaded PI PID Loop. Range: $\pm 3.4e^{+38}$
q_xLimActvPI	BOOL	TRUE: Output of the PI outer loop is being limited FALSE: Output from PI outer loop not being limited
q_xLimActvPID	BOOL	TRUE: Output of the PID inner loop is being limited FALSE: Output from PID inner loop not being limited
q_xErrPI	BOOL	Detected error in PI loop
q_uiErrIdPI	UNIT	Displays the detected error Id for the PI loop when q_xErrPI becomes TRUE Range: 0...4
q_xErrPID	BOOL	Detected error in PID loop
q_uiErrIdPID	UNIT	Displays the detected error Id for the PID loop when q_xErrPID becomes TRUE Range: 0...4

q_uiErrIdPI

This unique integer value indicates some particular detected error:

Detected Error ID	Description
0	No error is detected
1	i_stParaPI.tCyclTime out of range
2	i_stParaPI.rMaxLim < i_stParaPI.rMinLim
3	i_stParaPI.rKp less than zero
4	i_stParaPI.tTn out of range

q_uiErrIdPID

This unique integer value indicates some particular detected error:

Detected Error ID	Description
0	No error is detected
1	i_stParaPID.tTv out of range or i_stParaPID.tTn out of range or i_stParaPID.tTd out of range or i_stParaPID.rKp less than zero.
2	i_stParaPID.tTd < (i_stParaPI.tCyclTime / 2)
3	i_stParaPID.rMaxLim < i_stParaPID.rMinLim
4	i_stParaPID.rOterWndo<i_stParaPID.rInerWndo or i_stParaPID.rOterWndo<0 or i_stParaPID.rInerWndo<0

Part IV
Equipment Control Functions

Overview

This part describes the functionality and implementation of equipment control function blocks.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
12	FB_Cyclic_Monitoring: Cyclic Monitoring	95
13	FB_DeadBand: Suppressing Amplitude Oscillations	101
14	FB_Limiter: Limiting Input Signals	105
15	FB_PWM: Providing a PWM Output	109
16	FB_Redundant_Sensor_Monitoring: Redundant Sensor Monitoring	119
17	FB_Scaling: Scaling Input Signals	127
18	FB_Sensor_Monitoring: Sensor Monitoring	131

Chapter 12
FB_Cyclic_Monitoring: Cyclic Monitoring

Overview

This chapter explains the `FB_Cyclic_Monitoring` function block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
`FB_Cyclic_Monitoring` Function Block	96
Input Pin Description	98
Output Pin Description	98
Instantiation and Usage Example	99

FB_Cyclic_Monitoring Function Block

Pin Diagram

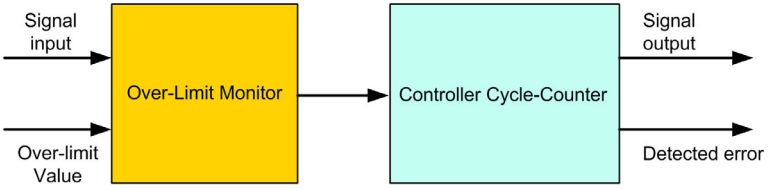
This figure shows the pin diagram of the FB_Cyclic_Monitoring function block:



Functional Description

The FB_Cyclic_Monitoring function block monitors an input signal for a maximum value (percentage of the absolute maximum value), over a pre-defined number of controller cycles before an inoperable over-limit is detected.

This function block is used to monitor a real input signal and to transfer the input signal to the output only if the input is within limits. It causes the input value to remain above a predefined limiting value for more than a predefined number of consecutive cycles.

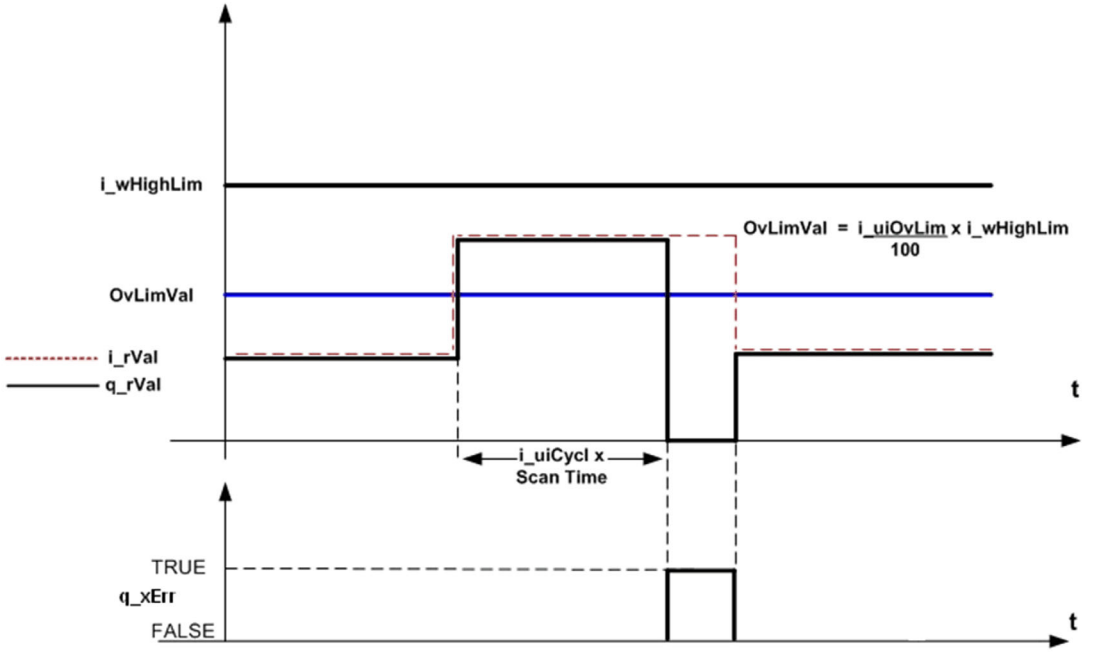


In normal operation, the input signal is transferred to the output based on following conditions

- If the input signal is less than or equal to over-limit (%) of high limit.
- If the input signal is exceeding the high limit for “n” number of consecutive cycle which is less than cycle input.

Timing Diagram

This figure shows the timing diagram for the FB_Cyclic_Monitoring function block:



Detected Error State

If the input signal is exceeding over-limit (%) of high limit for n number of cycle which is greater or equal to cycle input, then output is set to zero and detected error output is set to TRUE. The detected error is reset automatically if the input signal is within the limit.

Input Pin Description

Input Pin Description

This table describes the input pins of the FB_Cyclic_Monitoring function block.

Input	Data Type	Description
i_wHighLim	WORD	High limit of signal Range: 0...65535
i_uiOvlim	UINT	% of i_wHighLim which defines the over-limit range. Range: 0...100 Input value exceeding 100% is limited to 100%.
i_rVal	REAL	The input signal to be monitored Range: ±3.4e^{+38}
i_uiCycl	UINT	The number of controller cycles the FB allows input to output when input exceeds the over limit range. Range: 0...65535

Output Pin Description

Output Pin Description

This table describes the output pins of the FB_Cyclic_Monitoring function block

Output	Data Type	Description
q_rVal	REAL	Monitored output value. Range: ±3.4e^{+38}
q_xErr	BOOL	Detected error bit
q_rPerc	REAL	Monitored input value (i_rVal) in % of i_wHighLim. Range: ±3.4e^{+38}

Instantiation and Usage Example

Instantiation and Usage Example

This figure shows an instance of the the `FB_Cyclic_Monitoring` function block:



Example

This table shows and example for the `FB_CyclicMonitoring` function block operation:

Example	Inputs	Outputs	Remarks
1	`i_wHighLim` = 200, `i_uiOvLim` = 50, `i_rVal` = 40, `i_uiCycle` = 10	`q_rVal` = 40, `q_rPerc` = 20, `q_xErr` = FALSE	Input value (`i_rVal`) is less than or equal to calculated over limit value (That is [(`i_uiOvLim/100`) x `i_wHighLim`]). Output: Input `i_rVal` is assigned to output `q_rVal` & `q_xErr` output is FALSE.
2	`i_wHighLim` = 200, `i_uiOvLim` = 50, `i_rVal` = 110, `i_uiCycle` = 10	`q_rVal` = 110 `q_rPerc` = 55 `q_xErr` = FALSE	Input value (`i_rVal`) is greater than calculated over limit value (That is [(`i_uiOvLim/100`) x `i_wHighLim`]) and, **completed scan cycles are less than set no of cycles**. Output: Input `i_rVal` is assigned to output `q_rVal` & `q_xErr` output is FALSE.
3	`i_wHighLim` = 200, `i_uiOvLim` = 50, `i_rVal` = 110, `i_uiCycle` = 10	`q_rVal` = 0 `q_rPerc` = 55 `q_xErr` = TRUE	Input Value (`i_rVal`) is greater than calculated over limit value (That is [(`i_uiOvLim/100`) x `i_wHighLim`)] and **completed scan cycles are equal to or greater than set no of cycles**. Output: Output `q_rVal` is equal to zero and `q_xErr` output is TRUE.

Detected Error state

This table shows some general issues and their solution:

Issue	Cause	Solution
Detected error state	Input value(`i_rVal`) is not within the over limit (%) of `i_wHighLim` for n number of cycles	Input value (`i_rVal`) less than Over Limit value automatically resets detected error.

Chapter 13
FB_DeadBand: Suppressing Amplitude Oscillations

Overview

This chapter describes the FB_DeadBand function block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
FB_DeadBand Function Block	102
Input Pin Description	104
Output Pin Description	104

FB_DeadBand Function Block

Pin Diagram

This figure shows the pin diagram of the FB_DeadBand function block:



Functional Description

The FB_DeadBand function block is a Deadband function block which allows the input to pass on to the output only if the input is greater than the Deadband limit.

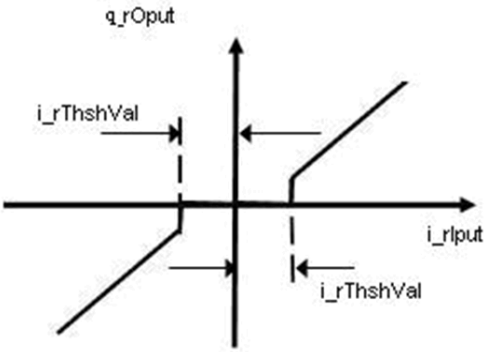
This function block suppresses small amplitude oscillations that are caused by noise, quantization or parameter calculation. It suppresses an input signal if it is within the threshold as shown in transfer function figure below.

With reference to the timing diagram:

- If the i_rIput is lower than the defined threshold range, q_rOput is set to zero and q_xSigUndThsh is TRUE.
- If the input value (i_rIput) greater or equal to the threshold range, q_rOput is equal to the i_rIput value.

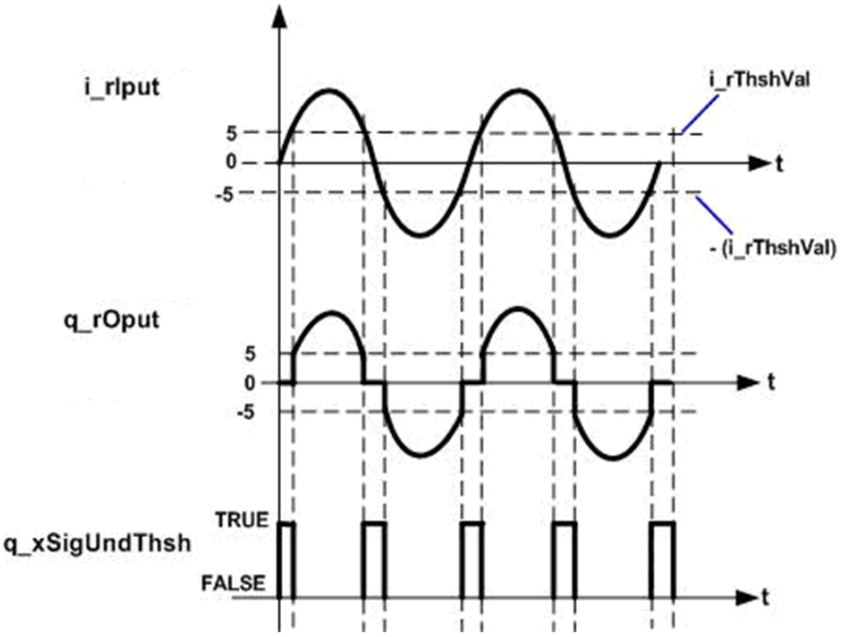
The q_xEn is TRUE as long as i_xEn is TRUE regardless of the detected error.

This figure shows the transfer function for FB_DeadBand function block:



Timing Diagram

This figure shows the timing diagram for FB_DeadBand function block:



Detected Error State

An invalid parameter at the function block inputs results in a detected error and corresponding detected error ID is generated. During the error detected state, the output value is set to zero.

The detected error can be reset only through rising edge of `i_xErrRst` input.

The `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Description

This table describes the input pins of the FB_DeadBand function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block. FALSE: Disables the function block.
i_rIput	REAL	Input value Range: ±3.4e^{+38}
i_rThshVal	REAL	Threshold value Range: 0.0...3.4e^{+38}
i_xErrRst	BOOL	Reset for detected error (on rising edge) (Optional)

Output Pin Description

Output Pin Description

This table describes the output pins of the FB_DeadBand function block.

Output	Data Type	Description
q_xEn	BOOL	TRUE if function block is enabled.
q_xBusy	BOOL	TRUE if function block is enabled and no detected error.
q_rOput	REAL	Output of the given input. Range: ±3.4e^{+38}
q_xSigUndThsh	BOOL	TRUE if input is under threshold limit.
q_xErr	BOOL	Detected error
q_uiErrId	UINT	0 = No detected error 1 = Invalid parameter i_rThsh < 0 Range: 0...1

Chapter 14
FB_Limiter: Limiting Input Signals

Overview

This chapter describes the `FB_Limiter` function block.

What Is in This Chapter?

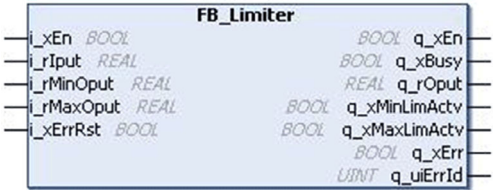
This chapter contains the following topics:

Topic	Page
`FB_Limiter` Function Block	106
Input Pin Description	108
Output Pin Description	108

FB_Limiter Function Block

Pin Diagram

This figure shows the pin diagram of the FB_Limiter function block:



Functional Description

The FB_Limiter function block is a limiter function block to limit an input signal within a defined range.

The input signal is limited to a defined range based on the i_rMaxOput and i_rMinOput as shown in the transfer function figure below.

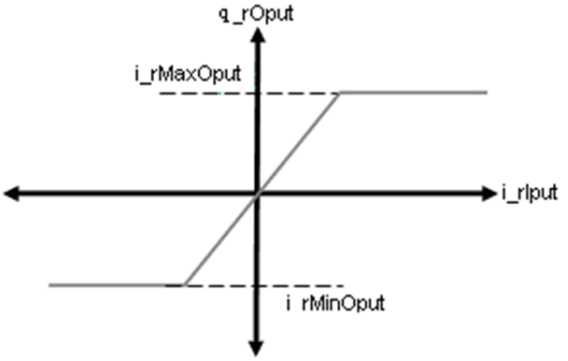
If the input exceeds the upper or lower limit, the output is limited to maximum or minimum values respectively.

With reference to the timing diagram below:

- If the input is within the defined range, the output is equal to input value.
- If input value exceeds the maximum limit, the output is limited to maximum output value.
- Similarly, if the input goes below the minimum output value, the output is limited to the minimum output value.
- If the function block limits the output, then q_xMinLimActv or q_xMaxLimActv is TRUE, based on the type of limit.

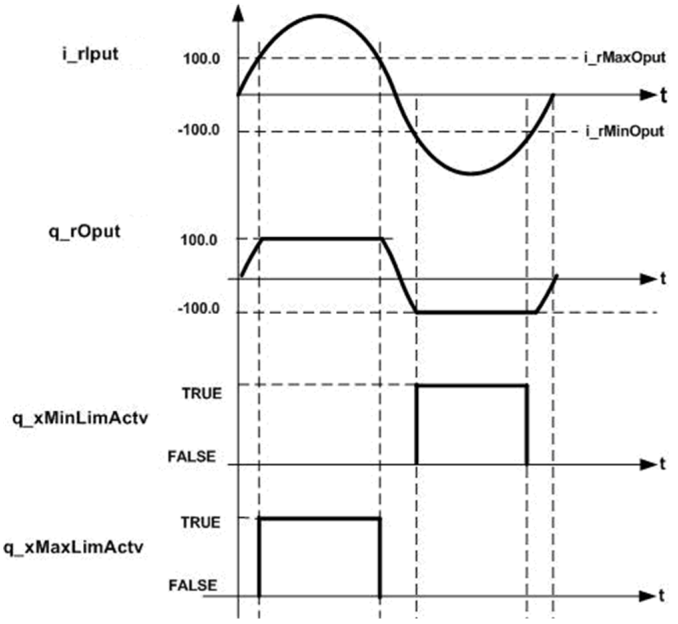
The q_xEn is TRUE as long as i_xEn is TRUE, regardless of detected error.

This figure shows the transfer function of the `FB_Limiter` function block:



Timing Diagram

This figure shows the timing diagram of the `FB_Limiter` function block:



Detected Error State

An invalid parameter at the function block inputs results in detected error, and a corresponding detected error ID will be generated.

During a detected error state, the output will be set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input. The output `q_xBusy` is TRUE whenever the function block is enabled and there is no detected error.

Input Pin Description

Input Pin Description

This table describes the input pins of the `FB_Limiter` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: Enables the function block. FALSE: Disables the function block.
`i_rIput`	`REAL`	Input value which has to be limited. Range: $\pm 3.4e^{+38}$
`i_rMinOput`	`REAL`	Minimum output value. Range: $\pm 3.4e^{+38}$
`i_rMaxOput`	`REAL`	Maximum output value. Range: $\pm 3.4e^{+38}$
`i_xErrRst`	`BOOL`	Reset for detected error (on rising edge) (Optional)

Output Pin Description

Output Pin Description

This table decscribes the input pins of the `FB_Limiter` function block:

Output	Data Type	Description
`q_xEn`	`BOOL`	TRUE if function block is enabled.
`q_xBusy`	`BOOL`	TRUE if function block is enabled and no detected error.
`q_rOput`	`REAL`	Output of the given input. Range: $\pm 3.4e^{+38}$
`q_xMinLimActv`	`BOOL`	TRUE if input is equal or under the minimum output value.
`q_xMaxLimActv`	`BOOL`	TRUE if input is equal or above maximum output value.
`q_xErr`	`BOOL`	Detected error
`q_uiErrId`	`UINT`	0 = No detected error 1 = Invalid parameter `i_rMaxOput` < `i_rMinOput` Range: 0 ... 1

Chapter 15
FB_PWM: Providing a PWM Output

Overview

This chapter describes the FB_PWM function block.

What Is in This Chapter?

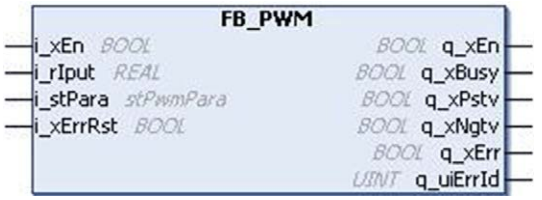
This chapter contains the following topics:

Topic	Page
FB_PWM Function Block	110
Input Pin Description	116
Structure Used	116
Output Pin Description	117

FB_PWM Function Block

Pin Diagram

This figure shows the pin diagram of the FB_PWM function block:



Functional Description

The FB_PWM function block is developed to provide a PWM output based on the input parameter.

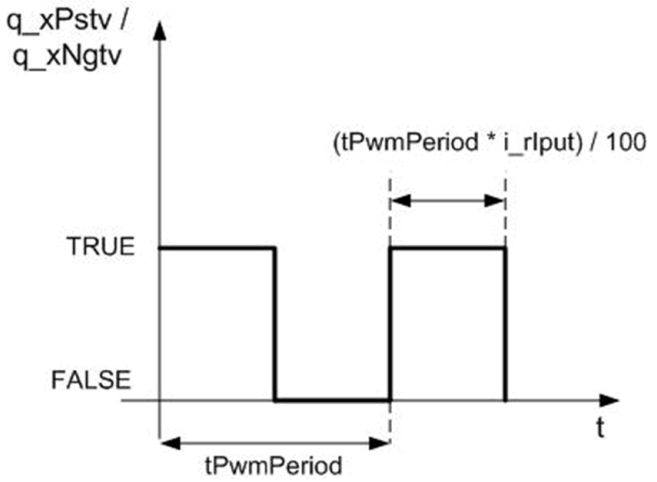
The PWM output is generated with defined ON time and OFF time as per the input shown in the first timing diagram below.

With reference to the second timing diagram:

- If i_rIput is a positive value, then the PWM output is available in the q_xPstv. The input i_rIput should be in a range of -100 to 100. The ON time of PWM is determined as given below: PWM ON time = (i_rIput x tPwmPeriod) / 100.
- If i_rIput is a negative value, then the PWM output is available in q_xNgtv.
- If i_rIput is greater than 100, then it is limited to 100 and if i_rIput is less than –100, then it is limited to –100.
- If i_xPwmInstUpdt is TRUE, the change in input parameter is updated in the current PWM cycle itself as shown in the timing diagram.
- If i_xPwmInstUpdt is FALSE, the change in input is updated only during a start of a new PWM cycle.

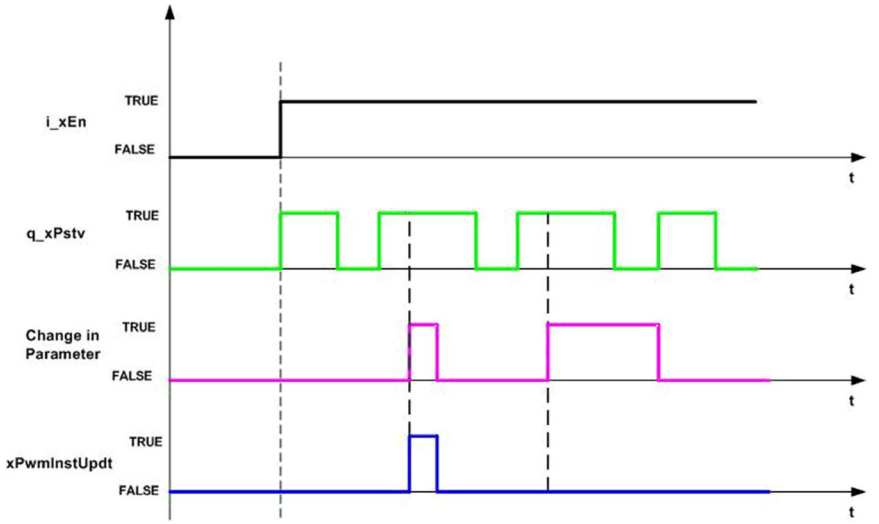
The q_xEn is TRUE as long as the input i_xEn is TRUE, regardless of detected error.

This figure shows the timing diagram for FB_PWM calculation:



Timing Diagram

This figure shows the timing diagram for FB_PWM function block:

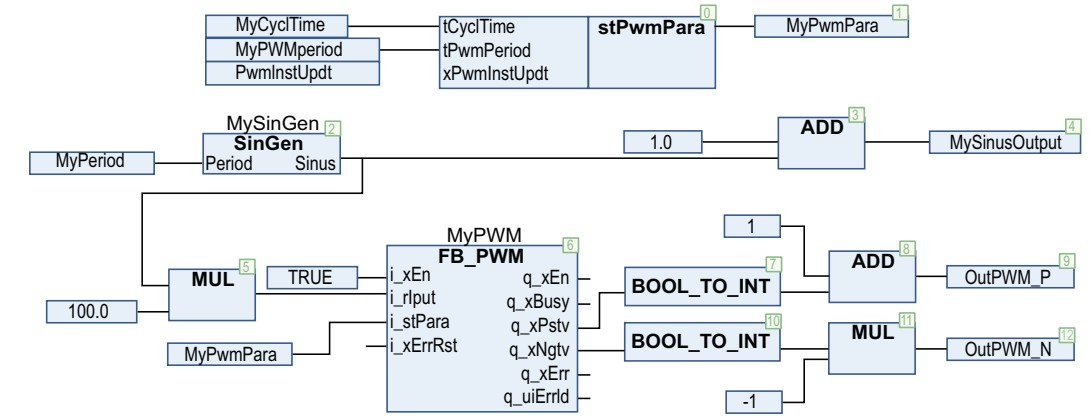


Example with a Frequency Signal

The program creates a Sinus signal at a certain period (5 seconds/0.2 Hz). This Sinus signal is the input of the FB_PWM.

MyPou_PWM [MyController: PLC Logic: Application] | Trace_PWM [MyController: PLC Logic: Application]

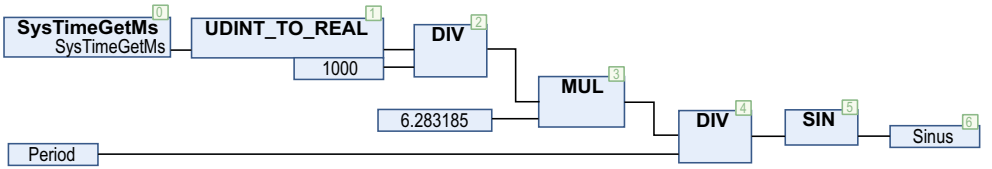
```
PROGRAM MyPou_PWM
VAR
    FlagDemarrer        : BOOL;
    MyPWM               : FB_PWM;
    MyPwmPara           : stPwmPara;
    MySinGen            : SinGen;
    MyCyclTime          : TIME := T#10MS;
    MyPWMperiod         : TIME := T#200MS;
    PwmInstUpdt         : BOOL;
    MyPeriod            : REAL := 5.0;
    MySinusOutput       : REAL;
    OutPWM_P            : INT;
    OutPWM_N            : INT;
END_VAR
```



Definition of the SinGen function block:

MyPou_PWM [MyController: PLC Logic: Application] | Trace_PWM [MyController: PLC Logic: Application] | SinGen [MyContro...

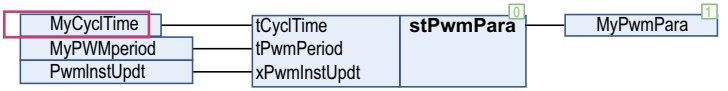
```
FUNCTION_BLOCK SinGen
VAR_INPUT
    Period : REAL;
END_VAR
VAR_OUTPUT
    Sinus : REAL;
END_VAR
VAR
END_VAR
```

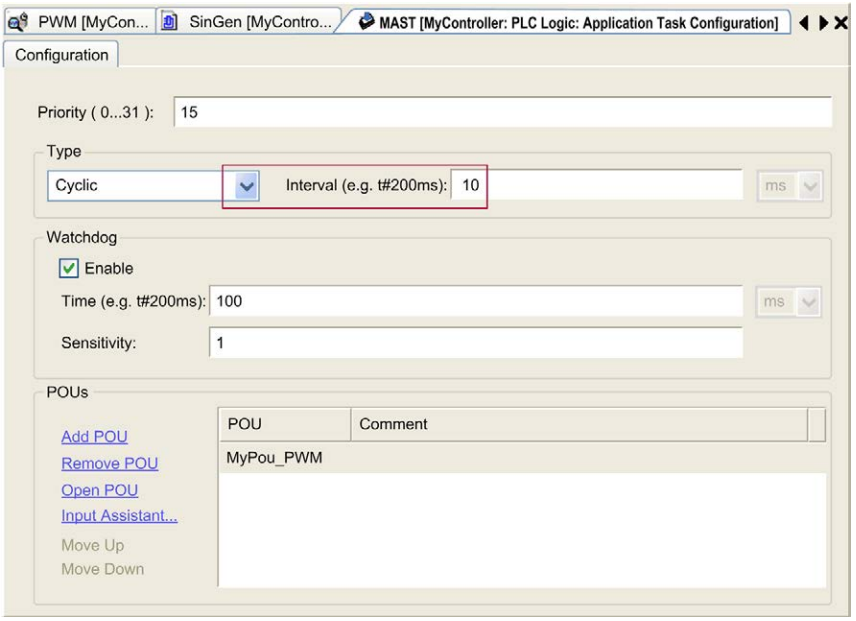


The input stPwmPara.tCycTime of the FB_PWM function block must have exactly the same value as the period of the POU in the MAST, here 10 milliseconds (see the red bordered area).

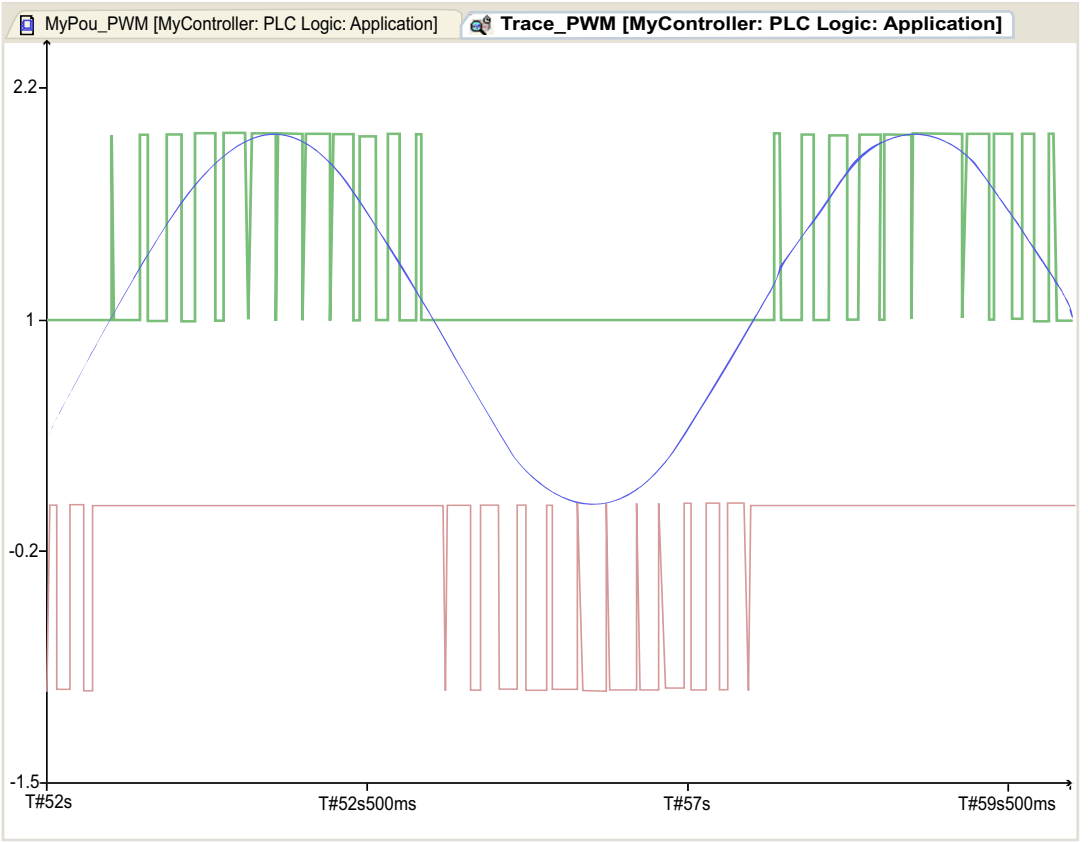
MyPou_PWM [MyController: PLC Logic: Application] | Trace_PWM [MyController: PLC Logic: Application]

```
PROGRAM MyPou_PWM
VAR
    FlagDemarrer        : BOOL;
    MyPWM               : FB_PWM;
    MyPwmPara           : stPwmPara;
    MySinGen            : SinGen;
    MyCyclTime          : TIME := T#10MS;
    MyPWMperiod         : TIME := T#200MS;
    PwmInstUpdt         : BOOL;
    MyPeriod            : REAL := 5.0;
    MySinusOutput       : REAL;
    OutPWM_P            : INT;
    OutPWM_N            : INT;
END_VAR
```




PWM [MyCon...
SinGen [MyContro...
MAST [MyController: PLC Logic: Application Task Configuration]
Configuration
Priority (0...31):
15
Type
Cyclic
Interval (e.g. t#200ms):
10
ms
Watchdog
Enable
Time (e.g. t#200ms):
100
ms
Sensitivity:
1
POUs
Add POU
Remove POU
Open POU
Input Assistant...
Move Up
Move Down
POU
Comment
MyPou_PWM

The result of the previous POU:



Blue `i_rIput` sinus signal at 0.2 Hz (function block `My_Filter_PT1_1`).
Green `q_xPstv` (an offset is added for the trace).
Red `q_xNgtv` (the signal is inversed for the trace).

Detected Error State

An invalid parameter at the function block inputs results in detected error and corresponding detected error ID is generated.

During detected error state, the output is set to zero.

The detected error can be reset only through rising edge of `i_xErrRst` input. The output `q_xBusy` is TRUE whenever the function block is enabled and there is no detected error.

Input Pin Description

Input Pin Description

This table shows the input pins of the FB_PWM function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block. FALSE: Disables the function block.
i_rIput	REAL	PWM Input value Range: -100...100
i_stPara	STRUCT stPwmPara	Structure parameter
i_xErrRst	BOOL	Reset for detected error (on rising edge) (Optional)

Structure Used

stPwmPara

Structure Element	Type	Description
tCyclTime	TIME	Task cycle time Range: 1...1e^{32} ms
tPwmPeriod	TIME	PWM time period Range: 1...1e^{32} ms
xPwmInstUpdt	BOOL	TRUE: a new input value is immediately adopted, even in the present PWM cycle. (Optional)

Output Pin Description

Output Pin Description

Output Pin Description

This table describes the input pins of the FB_PWM function block:

Output	Data Type	Description
q_xEn	BOOL	TRUE if function block is enabled.
q_xBusy	BOOL	TRUE if function block is enabled and no detected error.
q_xPstv	BOOL	PWM output is positive if PWM input >0.
q_xNgtv	BOOL	PWM output is negative if PWM Input <0.
q_xErr	BOOL	Detected error
q_uiErrId	UINT	0 = No detected error 1 = Invalid parameter i_ tCyclTime = 0 2 = Invalid parameter i_ tPwmPeriod <= i_tCyclTime Range: 0...2

Chapter 16
FB_Redundant_Sensor_Monitoring: Redundant Sensor Monitoring

Overview

This chapter describes the `FB_Redundant_Sensor_Monitoring` function block.

What Is in This Chapter?

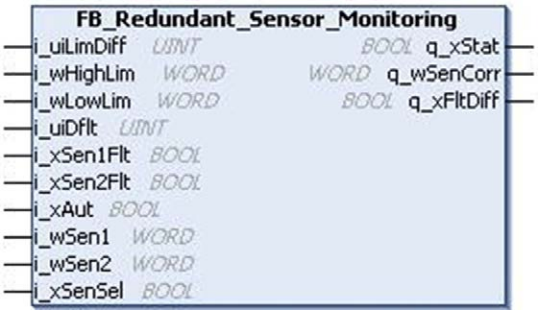
This chapter contains the following topics:

Topic	Page
`FB_Redundant_Sensor_Monitoring` Function Block	120
Input Pin Description	122
Output Pin Description	125

FB_Redundant_Sensor_Monitoring Function Block

Pin Diagram

This figure shows the pin diagram of the FB_Redundant_Sensor_Monitoring function block:



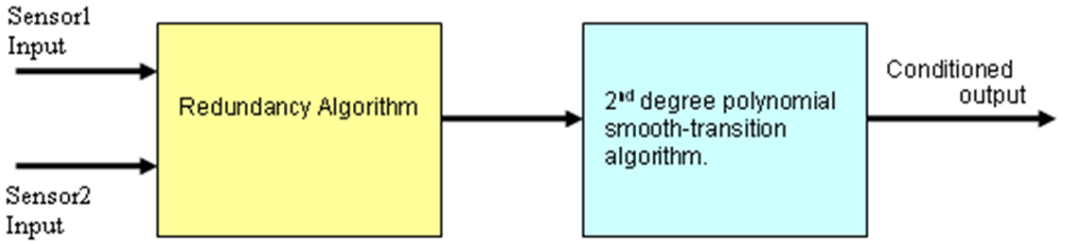
Functional Description

The FB_Redundant_Sensor_Monitoring function block monitors signals coming in from two redundant analog signal sources or field sensors which have the same range and characteristics.

The function block does the following functions:

- Manipulates the output as per sensor(s) readings (average value).
- Monitors if the 2 sensor readings are within the specified difference limit, otherwise takes corrective action.
- Performs predefined action in case of any inoperable sensor is reported.
- Automatic selection of healthy sensor for output in case of other inoperable sensor.
- Provision to select any sensor manually.
- A second-degree polynomial based 'smooth-transition algorithm' eliminates any sudden step variation in the output of the block.
- The output for a detected difference between the two sensors is set to TRUE when the difference between the two sensor values is not within the limits. The difference between sensor input values is reset immediately after the difference between the sensor inputs are within the limits.

This figure shows the block diagram of the `FB_Redundant_Sensor_Monitoring` function block:



This table contains the second degree polynomial smooth-transition algorithm conditions:

Example	Condition	FB Output
1	Absolute difference between calculated output and previous FB output is greater than or equal to 15, calculated output is greater than previous FB output.	Previous FB output - ((0.07 * (Calculated output - Previous FB output)) - 23)
2	Absolute difference between calculated output and previous FB output is greater than or equal to 15, calculated output is less than or equal to previous FB output.	Previous FB output - ((0.07 * (Calculated output - Previous FB output)) + 23)
3	Absolute difference between calculated output and previous FB output is less than 15.	Calculated output

Input Pin Description

Input Pin Description

This table describes the input pins of the FB_Redundant_Sensor_Monitoring function block:

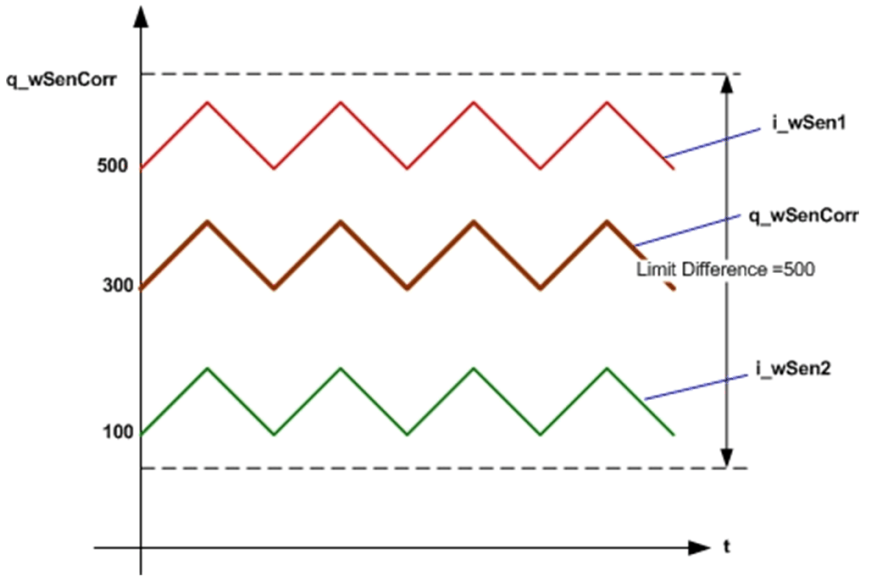
Output	Data Type	Description
i_uiLimDiff	UINT	The difference between sensor-1 and sensor-2 values in % of the sensor range within which the redundant sensors shall work. Range: 0...100 Value exceeding 100% is limited to 100%. 0%: There should not be any difference between sensors. 100%: Accept any value that comes in or always average Sen1_ip and Sen2_ip.
i_wHighLim	WORD	High limit of sensor inputs Range: 0...65535
i_wLowLim	WORD	Low limit of sensor inputs Range: 0...65535
i_uiDflt	UINT	Default value in terms of % of the sensor input range Range: 0...100
i_xSen1Flt	BOOL	TRUE: Sensor is in detected error FALSE: Sensor is healthy. (Optional)
i_xSen2Flt	BOOL	TRUE: Sensor is in detected error. FALSE: Sensor is healthy. (Optional)
i_xAut	BOOL	TRUE: Auto mode FALSE: Manual mode.
i_wSen1	WORD	Sensor-1 raw value Range: 0...65535 (It is expected that sensor-1 and 2 shall be of same range and characteristics.)
i_wSen2	WORD	Sensor-2 raw value Range: 0...65535 (It is expected that sensor-1 and 2 shall be of same range and characteristics.)
i_xSenSel	BOOL	TRUE: Sensor 2 selected FALSE: Sensor 1 selected Applicable only for manual mode.

i_uiLimDiff

This is Limit difference in terms of %. Averaged output (q_wSenCorr) is generated only if the difference between two sensor inputs is less than the Limit Difference.

- Limit difference is calculated based on the below equation
  Limit Difference = (i_uiLimDiff x (i_wHighLim - i_wLowLim))/100
- If the difference between Sensor 1 input and sensor 2 input is less than the Limit Difference, then output is average of two sensors as shown in the following figure.

This figure shows the averaging function in FB_Redundant_Sensor_Monitoring function block:

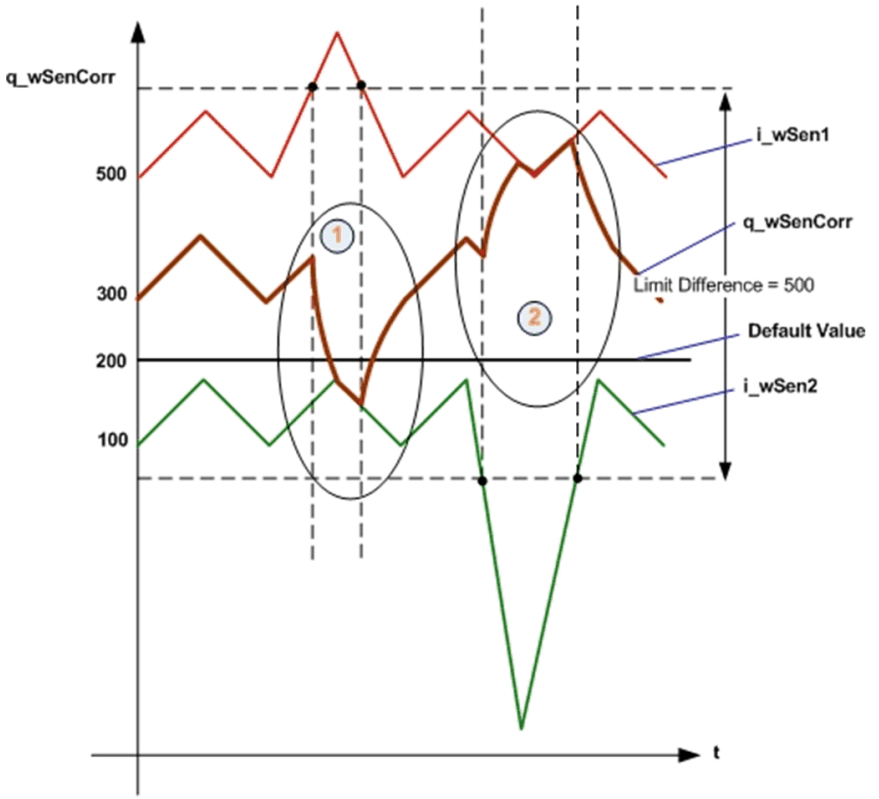


i_uiDflt

This input is default value in terms of % which is used to generate most appropriate output the difference is not within the limit and both the sensors are healthy.

- Default value is calculated based on the below equation:
  Default value = (i_uiDflt x (i_wHighLim - i_wLowLim))/100
- If the difference between 2 sensor input is out of limit, the function block gives an output of anyone sensor which is closer to the default value as shown in the following figure.

This figure shows the default value function in `FB_Redundant_Sensor_Monitoring` function block:



1. Difference between the sensors is not within the limit & Sensor 2 input is closer to default value, so Output follows the Sensor 2 regardless of sensor1 This transition is gradual due to Smooth transition algorithm.
2. Difference between the sensors is not within the limit & Sensor 1 input is closer to default value, so Output follows the Sensor 1 regardless of sensor 2. This transition is gradual due to Smooth transition algorithm.

i_xSen1Flt and i_xSen2Flt

These inputs are used to detect whether the two sensors are healthy or not healthy.

- If both the sensors are not healthy, the output (q_wSenCorr) is set to zero.
- If sensor 1 is not healthy, then the output is set to sensor 2 input. Similarly if sensor 2 is not healthy, then the output is set to sensor 1 input.

i_xAut

This input is used to select auto mode or manual mode.

- If this input is TRUE, the function block operates in auto mode. In auto mode, based on sensor inputs and difference between the inputs, function block generates an appropriate output.
- If this input is FALSE, the function block operates in manual mode. In manual mode, based i_xSenSel input the output is forced to either sensor 1 or sensor 2.
- If i_xSenSel is FALSE, sensor 1 input is selected. Similarly if i_xSenSel is TRUE, sensor 2 input is selected.

Output Pin Description

Output Pin Description

This table describes the output pins of the FB_Redundant_Sensor_Monitoring function block:

Output	Data Type	Description
q_xStat	BOOL	TRUE: Auto mode FALSE: Manual mode
q_wSenCorr	WORD	Redundant sensor manipulated value Range: 0...65535
q_xFltDiff	BOOL	TRUE: Detected error FALSE: Normal This output is TRUE when difference between the sensors is not within limit for more than 3 consecutive controller cycles in auto mode.

NOTE: The function block output q_wSenCorr will not change abruptly based on input. Instead a second-degree polynomial is applied to avoid such sudden step variation in the output of the function block.

Chapter 17
FB_Scaling: Scaling Input Signals

Overview

This chapter describes the FB_Scaling function block.

What Is in This Chapter?

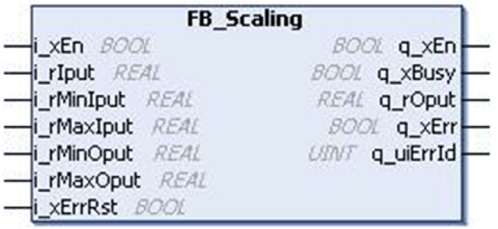
This chapter contains the following topics:

Topic	Page
FB_Scaling Function Block	128
Input Pin Description	130
Output Pin Description	130

FB_Scaling Function Block

Pin Diagram

This figure shows the pin diagram of the FB_Scaling function block:

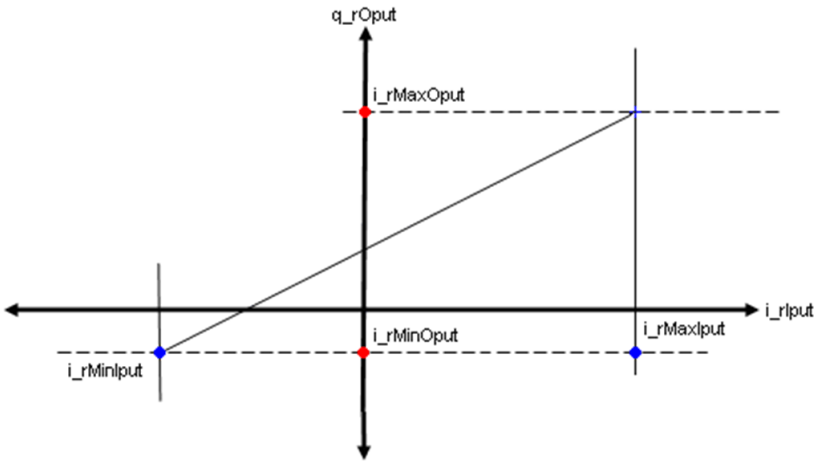


Functional Description

The FB_Scaling function block is developed to convert an input value to a specified output range linearly and an error is detected in case of an invalid parameter.

This function block scales an input signal to a linear output, relative to a defined maximum and minimum range.

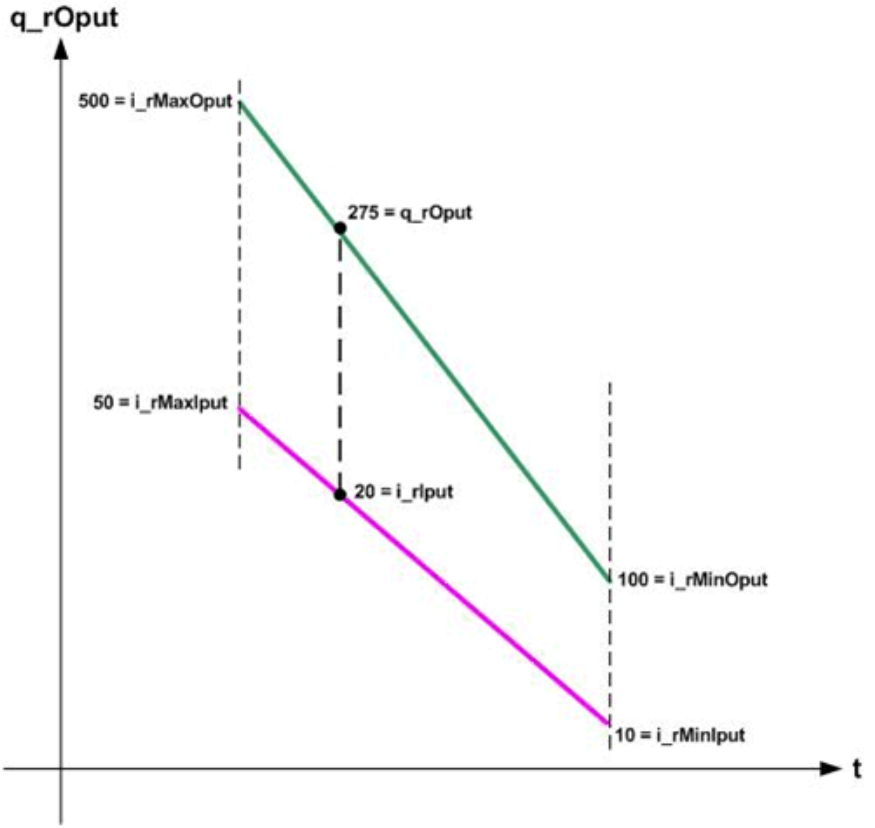
The input signal is scaled in a linear manner with reference to two value ranges as shown in the figure below:



The output changes dynamically based on the change in input:

- Slope = (`i_rMaxOput-i_rMinOput`) / (`i_rMaxIput` - `i_rMinIput`)
- Offset = `i_rOutMax` - (Slope * `i_rMaxIput`)
- `q_rOput` = (Slope * `i_rIput`) + Offset

For an `i_rIput` in the range of `i_rMinIput` and `i_rMaxIput`, `q_rOput` scaled to a range of `i_rMinOput` and `i_rMaxOput`. The `q_xEn` is TRUE as long as the input `i_xEn` is TRUE, regardless of a detected error, as shown in the figure:



Detected Error State

An invalid parameter at the function block inputs results in a detected error and a corresponding detected error ID is generated. The output is set to zero during a detected error. The detected error can be reset only through a rising edge of `i_xErrRst`. The input `q_xBusy` is TRUE whenever the function block is enabled and there is no detected error.

Input Pin Description

Input Pin Description

This table describes the input pins of the FB_Scaling function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: Enables the function block. FALSE: Disables the function block.
i_rIput	REAL	Input value which has to be scaled. Range: ±3.4e^{+38}
i_rMinIput	REAL	Minimum input value Range: ±3.4e^{+38}
i_rMaxIput	REAL	Maximum input value Range: ±3.4e^{+38}
i_rMinOput	REAL	Minimum output value Range: ±3.4e^{+38}
i_rMaxOput	REAL	Maximum output value Range: ±3.4e^{+38}
i_xErrRst	BOOL	Reset for detected error (on rising edge) (Optional)

Output Pin Description

Output Pin Description

The following table includes the different outputs of the function block along with the description of the identifiers or commands.

Output	Data Type	Description
q_xEn	BOOL	TRUE if function block is enabled.
q_xBusy	BOOL	TRUE if function block is enabled and no detected error.
q_rOput	REAL	Scaled output of the given input. Range: ±3.4e^{+38}
q_xErr	BOOL	Detected error
q_uiErrId	UINT	0 = No detected error 1 = Invalid parameter i_rMinIput = i_rMaxIput Range: 0...1

Chapter 18
FB_Sensor_Monitoring: Sensor Monitoring

Overview

This chapter describes the `FB_Sensor_Monitoring` function block.

What Is in This Chapter?

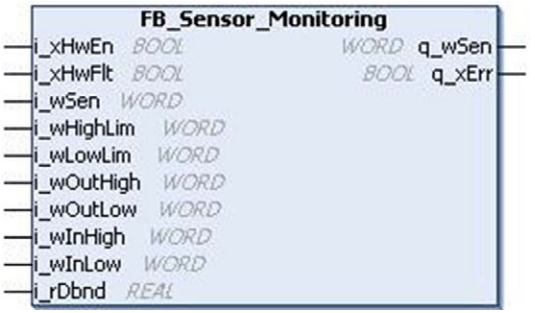
This chapter contains the following topics:

Topic	Page
`FB_Sensor_Monitoring` Function Block	132
Input Pin Description	133
Output Pin Description	134
Instantiation and Usage Example	134

FB_Sensor_Monitoring Function Block

Pin Diagram

This figure shows the pin diagram of the FB_Sensor_Monitoring function block:



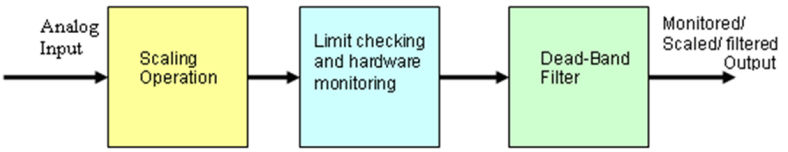
Functional Description

The FB_Sensor_Monitoring function block monitors and/or scales and/or Deadband filters an input analog signal.

This function block performs the following operations on an analog input signal:

- Monitor if the sensor reading is within the operator specified range, otherwise an error output is detected if not within the range.
- Monitor I/O hardware and generate an alarm if an error is detected.
- Provision to enable/disable the I/O hardware monitoring feature in the block.
- Scale the input value to the desired output range.
- Pass the final output through a dead-band filter. Deadband suppresses the relative oscillation based on the previous input and present input, and then generates an output.

This figure shows the FB_Sensor_Monitoring block diagram:



Input Pin Description

Input Pin Description

This table describes the input pins of the FB_Sensor_Monitoring function block:

Input	Data Type	Description
i_xHwEn	BOOL	TRUE: Hardware monitoring enabled FALSE: Hardware monitoring disabled
i_xHwFlt	BOOL	TRUE: Hardware detected error FALSE: No hardware detected error (Optional)
i_wSen	WORD	Sensor value Range: 0...65535
i_wHighLim	WORD	High limit of sensor input. Range: 0...65535 i_wHighLim should be greater than i_wLowLim
i_wLowLim	WORD	Low limit of sensor input. Range: 0...65535
i_wOutHigh	WORD	Output range high limit for scaling function. Range: 0..65535 For scaling function, i_wOutHigh, i_wOutLow, i_wInHigh & i_wInLow are used
i_wOutLow	WORD	Output range low limit for scaling function. Range: 0...65535
i_wInHigh	WORD	Input range high limit for scaling function. Range: 0...65535
i_wInLow	WORD	Input range low limit for scaling function. Range: 0...65535
i_rDbnd	REAL	Defines the width of the 'dead-band' in the DeadBand filter, in terms of % of range. Range: 0.0...100.0 0.0: No deadband filtering. 100.0: Block all signals. So ideally the value is less than 100.0 (Optional)

Output Pin Description

Output Pin Description

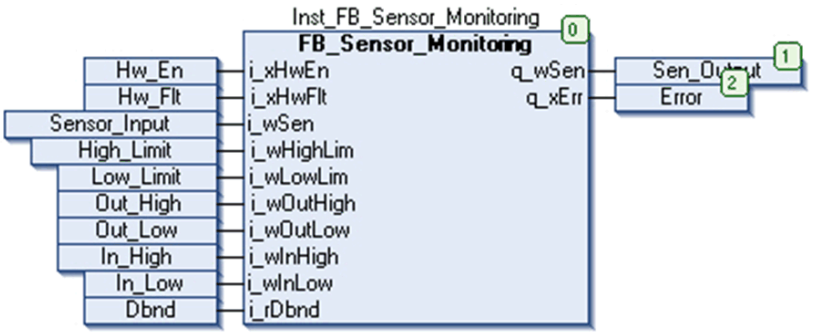
This table describes the output pins of the FB_Sensor_Monitoring function block:

Output	Data Type	Description
q_wSen	WORD	Valid sensor output Range: 0...65535
q_xErr	BOOL	TRUE: If there is a Hardware detected error or if the scaled output is not within the sensor limits for more than 3 consecutive controller scan cycles. FALSE: No detected error

Instantiation and Usage Example

Instantiation and Usage Example

This figure shows an instance of the FB_Sensor_Monitoring function block pin diagram:



Example

This example illustrates the functionality of different features in FB_Sensor_Monitoring function block:

Example	Steps	Inputs	Outputs
1	Scaling: Based on the scaling input parameters, sensor input i_wSen is scaled linearly and scaled output value is passed for limit checking.	i_wSen = 1000, i_wOutHigh = 2000, i_wOutLow = 100, i_wInHigh = 1000, i_wInLow = 10.	Internal calculated scaled output = 2000

Example	Steps	Inputs	Outputs
2	Limit checking: ● If calculated scaled output is greater than maximum limiting Input, i_wHighLim than output is limited to i_wHighLim ● If calculated scaled output is less than minimum limiting input, i_wLowLim than output is limited to i_wLowLim ● If calculated scaled output is within the limit, the calculated scaled output is processed further for deadband functions. q_xErr will be TRUE if calculated scaled output is out of range for more than 3 consecutive controller scan cycles.	i_wSen = 1000, i_wHighLim = 20000, i_wLowLim = 4000, i_wOutHigh = 2000, i_wOutLow = 100, i_wInHigh = 1000, i_wInLow = 10, i_rDbnd = 10.	Internal output after limit check = 4000 q_xErr = FALSE/TRUE.
3	Hardware detected error monitoring: q_xErr output is TRUE and q_wSen holds its last value when hardware detected error monitoring is enabled and hardware (i_xHwFlt) detected error input is TRUE.	i_xHwEn=1, i_xHwFlt=1, i_wSen=1000, i_wOutHigh=2000, i_wOutLow=100, i_wInHigh=1000, i_wInLow=10, i_rDbnd=10.	q_wSen = 4000, q_xErr = TRUE.
4	Deadband filtering: ● If difference between calculated scaled output and previous FB output is less or equal to calculated deadband difference value (That is [(i_rDbnd/100) x (i_wHighLim-i_wLowLim)]), final FB output is equal to previous FB output. ● If difference between calculated scaled output and previous FB output is greater than calculated deadband difference value (That is [(i_rDbnd/100) x (i_wHighLim-i_wLowLim)]), final FB output is equal to calculated scaled output. q_xErr status output can be equal to 0 or 1 depending on limit checking and hardware detected error monitoring functionality explained above. Note: i_rDbnd = 0: No deadband filtering. i_rDbnd >= 100: Block all signals.	i_wSen = 1000, i_wHighLim = 20000, i_wLowLim = 4000, i_wOutHigh = 2000, i_wOutLow = 100, i_wInHigh = 1000, i_wInLow = 10, i_rDbnd = 10.	q_wSen = 4000, q_xErr = False.

Part V
Filtering Functions

Overview

This part describes the functionality and implementation of filtering function blocks.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
19	Global Parameter List	139
20	Filter_AnalogInput: Checking Analog Input Varaibility	141
21	Filter_Arithmetic: Giving Arithmetic Mean Value	145
22	Filter_MovingAverage: Giving Moving Mean Value	149
23	Filter_PT1: Providing PT1 Transfer Function	153

Chapter 19
Global Parameter List

Global Parameter List (GPL)

Overview

Type:	Global parameters
Available as of:	3.0.1.0

Description

The global parameter list (GPL) contains global constants which are used by filtering function blocks of this library. The parameters can be edited individually for each application where the library is used. The modification must be done within the **Library Manager** of the project where the library is referenced.

Global Parameters

Variable	Data type	Default value	Range	Description
`Gc_uiMaxAvgeSmpl`	UINT	100	100... 1000	The maximum number of samples counted by the `i_uiSmplCnt` input *(see page 152)*.

Chapter 20
Filter_AnalogInput: Checking Analog Input Varaibility

Overview

This chapter describes the `Filter_AnalogInput` function block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
`Filter_AnalogInput` Function Block	142
Input Pin Description	144
Output Pin Description	144

Filter_AnalogInput Function Block

Pin Diagram

This figure shows the pin diagram of the Filter_AnalogInput function block:



Functional Description

The Filter_AnalogInput function block checks the plausibility on a measured analog input.

In normal state of operation, if the difference between the present and previous input value:

- is less than or equal to the specified value i_rMaxIputDiff, then the output follows the input value.
- is greater than the specified value i_rMaxIputDiff, then the output is overwritten by the previous output value for the maximum of three controller scan cycles. Output overwritten status bit q_xOputOvwr is TRUE in this condition.
- exceeds the specified value i_rMaxIputDiff for more than three consecutive controller scan cycles, the output again follows the input value.

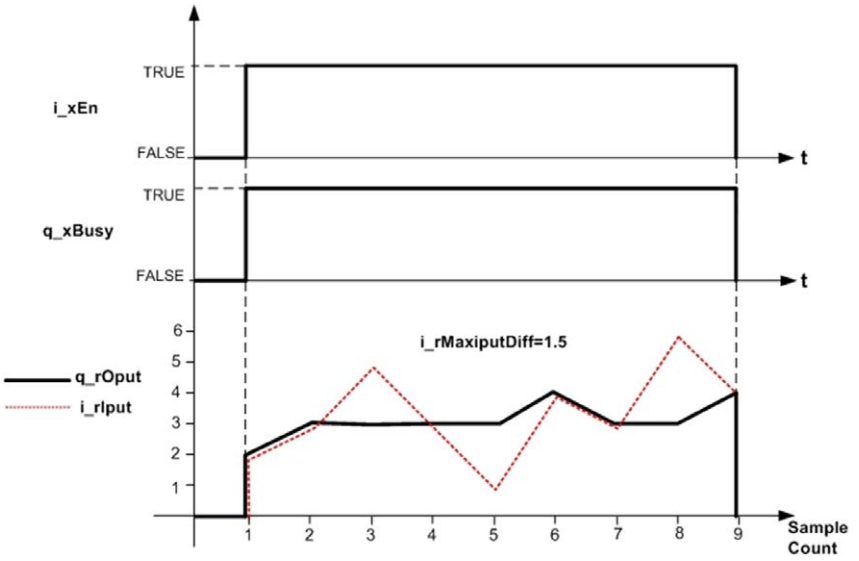
NOTE: When the function block is enabled, during the first scan cycle the input is assigned to the output.

Example

Maximum difference between present and previous inputs (i_rMaxIputDiff) = 1.5:

Scan Cycle	Input Value (i_rIput)	Output Value (q_rOput)	Output Overwritten Bit (q_xOputOvwr)
First	2.0	2.0	FALSE
Second	3.0	3.0	FALSE
Third	5.0	3.0	TRUE
Fourth	3.0	3.0	TRUE
Fifth	1.0	3.0	TRUE
Sixth	4.0	4.0	FALSE

This figure shows the normal behavior of the `Filter_AnalogInput` function block:



Detected Error State

Invalid parameter such as `i_rMaxIputDiff` < 0 results in a detected error and corresponding detected error ID is generated. During the error detected state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in the behavior of output figure above, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Description

This table describes the input pins of the `Filter_AnalogInput` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`i_rIput`	`REAL`	Analog input variable Range: $1.17e^{-38}$...$3.4e^{+38}$
`i_rMaxIputDiff`	`REAL`	Maximum difference between current and previous inputs for plausibility check Range: 0...$3.4e^{+38}$ `i_rMaxIputDiff` < 0 generates detected error
`i_xErrRst`	`BOOL`	TRUE: Reset the detected error (on rising edge). (Optional)

Output Pin Description

Output Pin Description

This table describes the output pins of the `Filter_AnalogInput` function block:

Output	Data Type	Description
`q_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`q_xBusy`	`BOOL`	TRUE: Active and no error detected. FALSE: Disabled or detected error.
`q_xOputOvwr`	`BOOL`	TRUE: If present output is overwritten by previous output. FALSE: If output follows present input.
`q_rOput`	`REAL`	Analog output value from Range: ±$3.4e^{+38}$.
`q_xErr`	`BOOL`	TRUE: Detected error. FALSE: No detected error.
`q_uiErrId`	`UINT`	Indicates the detected error number when detected error output is set. 0: No detected error. 1: Invalid parameter `i_rMaxIputDiff` < 0.

Chapter 21
Filter_Arithmetic: Giving Arithmetic Mean Value

Overview

This chapter describes the `Filter_Arithmetic` function block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
`Filter_Arithmetic` Function Block	146
Input Pin Description	148
Output Pin Description	148

Filter_Arithmetic Function Block

Pin Diagram

This figure shows the pin diagram of the Filter_Arithmetic function block:



Functional Description

The Filter_Arithmetic function block calculates the arithmetic average value for the user defined number of input samples.

Once the function block is enabled, the output calculation begins.

When the number of samples recorded is equal to the specified value i_uiSmplCnt, the function block gives an averaged output and the output valid bit q_xOputVld becomes TRUE.

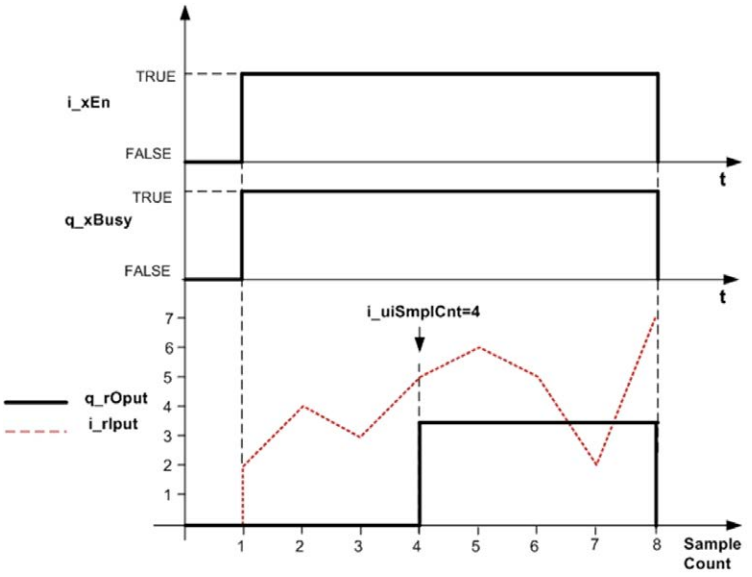
The function block output holds this value until the function block is disabled or is in the detected error state.

Example

Number of samples to average (i_uiSmplCnt) = 4:

Scan Cycle	Input Value (i_rIput)	Output Value (q_rOput)	Output Valid Bit (q_xOputVld)
First	2.0	0	FALSE
Second	4.0	0	FALSE
Third	3.0	0	FALSE
Fourth	5.0	3.5	TRUE
Fifth	6.0	3.5	TRUE
Sixth	5.0	3.5	TRUE

This figure shows normal output behavior:



Mathematical Background

This equation shows the generalized arithmetic mean value:

$$\bar{x} = \frac{1}{n}\left(\sum_{n} x_n\right)$$

Where:

n = Number of samples entered by user for mean value calculation,

Xn = Input samples,

$\bar{x}$ = Calculated output.

Detected Error State

Invalid parameter such as `i_uiSmplCnt` = 0 results in a detected error and corresponding detected error ID is generated. During the detected error state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in output behavior figure above, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Description

This table describes the input pins of the `Filter_Arithmetic` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`i_rIput`	`REAL`	Input value Range: ±3.4e^{+38}
`i_uiSmplCnt`	`UINT`	Number of samples (Input values) to average Range: 1...65535 `i_uiSmplCnt` = 0 generates detected error.
`i_xErrRst`	`BOOL`	TRUE: Reset the detected error (on rising edge). (Optional)

Output Pin Description

Output Pin Description

This table describes the output pins of the `Filter_Arithmetic` function bloc:

Output	Data Type	Description
`q_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`q_xBusy`	`BOOL`	TRUE: Active and no detected error. FALSE: Disabled or detected error.
`q_xOputVId`	`BOOL`	TRUE: If number of samples recorded is greater than or equal to `i_uiSmplCnt` input.
`q_rOput`	`REAL`	Output value from ±3.4e^{+38}.
`q_xErr`	`BOOL`	TRUE: Detected error. FALSE: No detected error.
`q_uiErrId`	`UINT`	Detected error number when error output is set. 0: No detected error. 1: Invalid parameter `i_uiSmplCnt` = 0.

Chapter 22
Filter_MovingAverage: Giving Moving Mean Value

Overview

This chapter describes the `Filter_MovingAverage` function block.

What Is in This Chapter?

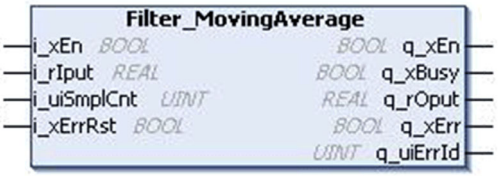
This chapter contains the following topics:

Topic	Page
`Filter_MovingAverage` Function Block	150
Input Pin Description	152
Output Pin Description	152

Filter_MovingAverage Function Block

Pin Diagram

This figure shows the pin diagram of the Filter_MovingAverage function block:



Functional Description

The Filter_MovingAverage function block calculates the moving average value for the user-defined number of input samples.

When the number of recorded samples are:

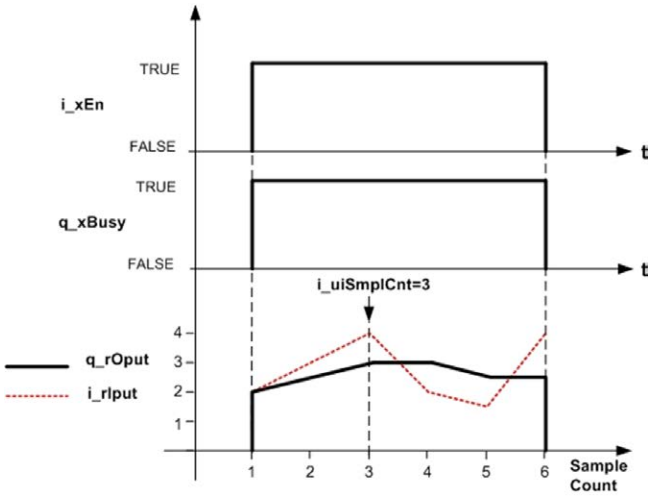
- Less than the specified value i_uiSmplCnt, the function block calculates the average value with the available number of inputs and gives the corresponding output.
- Equal to or greater than the specified value i_uiSmplCnt, the function block calculates the average value with i_uiSmplCnt number of inputs and gives the corresponding output. It operates like the moving average filter.
- For i_uiSmplCnt = 0, the input value is assigned to output.

Example

Number of Samples to average (i_uiSmplCnt) = 3:

Scan Cycle	Input Value (i_rIput)	Output Value (q_rOput)
1	2.0	2.0
2	3.0	2.5
3	4.0	3.0
4	2.0	3.0
5	1.5	2.5
6	4.0	2.5

This figure shows normal behavior:



Mathematical Background

This equation shows the generalized equation for the `Filter_MovingAverage` function:

$$\bar{x}_k^{\,n} = \frac{1}{n}\left(\sum_{i=k-n}^{k} x_i\right)$$

n = Number of samples,

x_i = Input samples,

k = `GPL.Gc_uiMaxAvgeSmpl`, internal constant,

$\bar{x}_k^{\,n}$ = Calculated output.

Note

In the event of a decrease in the number of sample counts (`i_uiSmplCnt`), the output (`q_rOput`) in the consequent scans is calculated by decreasing the number of samples by one in every consecutive scan.

Detected Error State

Invalid parameter such as `i_uiSmplCnt` > `GPL.Gc_uiMaxAvgeSmpl` results in detected error and corresponding detected error ID is generated.

During the detected error state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in behavior of output figure above, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Description

This table describes the input pins of the `Filter_MovingAverage` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`i_rIput`	`REAL`	Input value Range: ±3.4e^{+38}
`i_uiSmplCnt`	`UINT`	Number of samples (Input values) to average Range: 0...`GPL.Gc_uiMaxAvgeSmpl` `i_uiSmplCnt` > `GPL.Gc_uiMaxAvgeSmpl` generates detected error.
`i_xErrRst`	`BOOL`	Reset the detected error (on rising edge). (Optional)

Output Pin Description

Output Pin Description

This table describes the output pins of the `Filter_MovingAverage` function block:

Output	Data Type	Description
`q_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`q_xBusy`	`BOOL`	TRUE: Active and no detected error. FALSE: Disabled or detected error.
`q_rOput`	`REAL`	Output value Range: ±3.4e^{+38}
`q_xErr`	`BOOL`	TRUE: Detected error. FALSE: No detected error.
`q_uiErrId`	`UINT`	Detected error number when error output is set. 0: No detected error. 1: Invalid parameter `i_uiSmplCnt` > `GPL.Gc_uiMaxAvgeSmpl`.

Chapter 23
Filter_PT1: Providing PT1 Transfer Function

Overview

This chapter describes the `Filter_PT1` function block.

What Is in This Chapter?

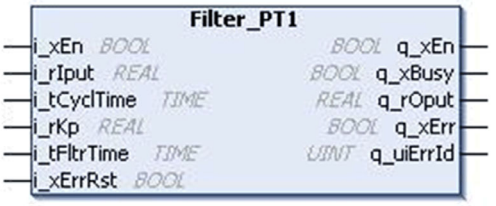
This chapter contains the following topics:

Topic	Page
`Filter_PT1` Function Block	154
Input Pin Description	157
Output Pin Description	157
Instantiation and Usage Example	158

`Filter_PT1` Function Block

Pin Diagram

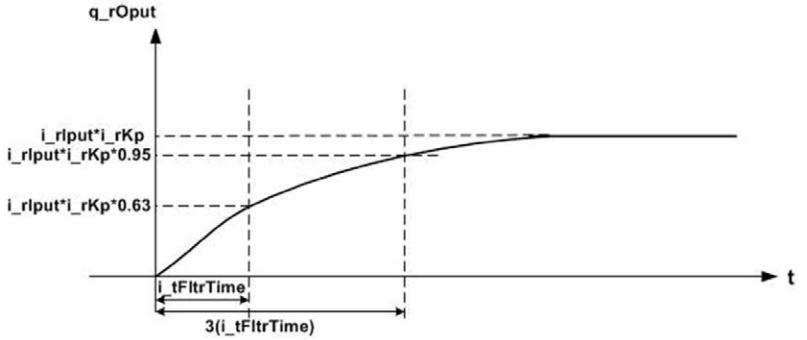
This figure shows the pin diagram of the `Filter_PT1` function block:



Functional Description

The `Filter_PT1` function block provides a PT1 transfer function. The output value increases to 63% of input value within the time period equal to filter time constant. The output value reaches to 95% of input value after the time period equal to 3 * Filter time constant and then reaches gradually to 100% of the input value.

This figure shows the output profile functionality of the `Filter_PT1` function block:



When the period is equal to:

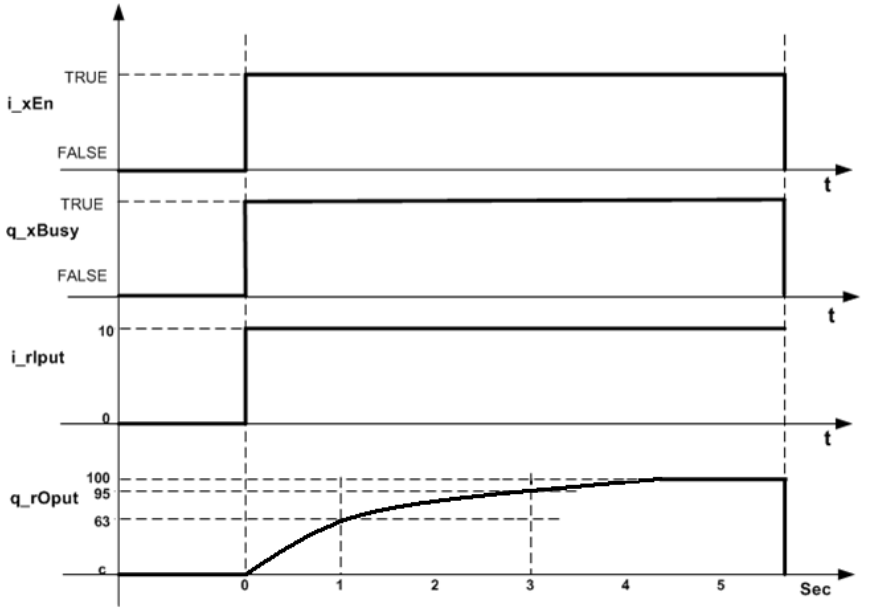
- The filter time constant, the output value increases to 63% of the input value,
- Three times the filter time constant, the output value increases to 95% of the input value and then gradually reaches to 100% of the input.

Example

If the input value (`i_rIput`) equals 10 and the filtering time constant (`i_tFltrTime`) is one second for a filtering gain of 10, then the output value (`q_rOput`) will be equal to 63 after a time period of one second.

The output value will be equal to 95 after a time period of three seconds (three times the filter time constant), and then the output will gradually reach to 100.

This figure shows normal behavior:



Mathematical Background

This equation shows the transfer function:

$$G(s) = K_p \frac{1}{1 + T_s}$$

Where:
Kp = Function PT1 gain or amplification
Ts = Function PT1 filter time constant
G(s) = transfer function

The equation shown above is a Laplace notation for the first-order low pass filter.

In digital-time systems, this function is often referred to as the pulse-transfer function (PT1 function).

Detected Error State

Invalid parameter such as `i_tCyclTime` = 0 or `i_tFltrTime` < `i_tCyclTime` results in a detected error and corresponding detected error ID is generated. During the detected error state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in function block the output figure, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

Input Pin Description

Input Pin Description

This table describes the output pins of the `Filter_PT1` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`i_rIput`	`REAL`	Input value for processing Range: ±3.4e^{+38}
`i_tCyclTime`	`TIME`	Task cycle time Range: 0...4294967295 ms `i_tCyclTime` < 0 generates detected error.
`i_rKp`	`REAL`	Function PT1 gain/amplification Range: ±3.4e^{+38}
`i_tFltrTime`	`TIME`	Filter time constant Range: 0...4294967295 ms Factory setting = t#0ms, `i_tFltrTime` < `i_tCyclTime` generates detected error.
`i_xErrRst`	`BOOL`	TRUE: Reset the detected error (On rising edge). (Optional)

Output Pin Description

Output Pin Description

This table describes the output pins of the `Filter_PT1` function block:

Output	Data Type	Description
`q_xEn`	`BOOL`	TRUE: Enabled FALSE: Disabled
`q_xBusy`	`BOOL`	TRUE: Active and no detected error. FALSE: Disabled or detected error.
`q_rOput`	`REAL`	Output value Range: ±3.4e^{+38}
`q_xErr`	`BOOL`	TRUE: Detected error. FALSE: No detected error.
`q_uiErrId`	`UINT`	Detected error number when error output is set. 0: No detected error. 1: Invalid parameter `i_tCyclTime` < 0. 2: Invalid parameter `i_tFltrTime` < `i_tCyclTime`.

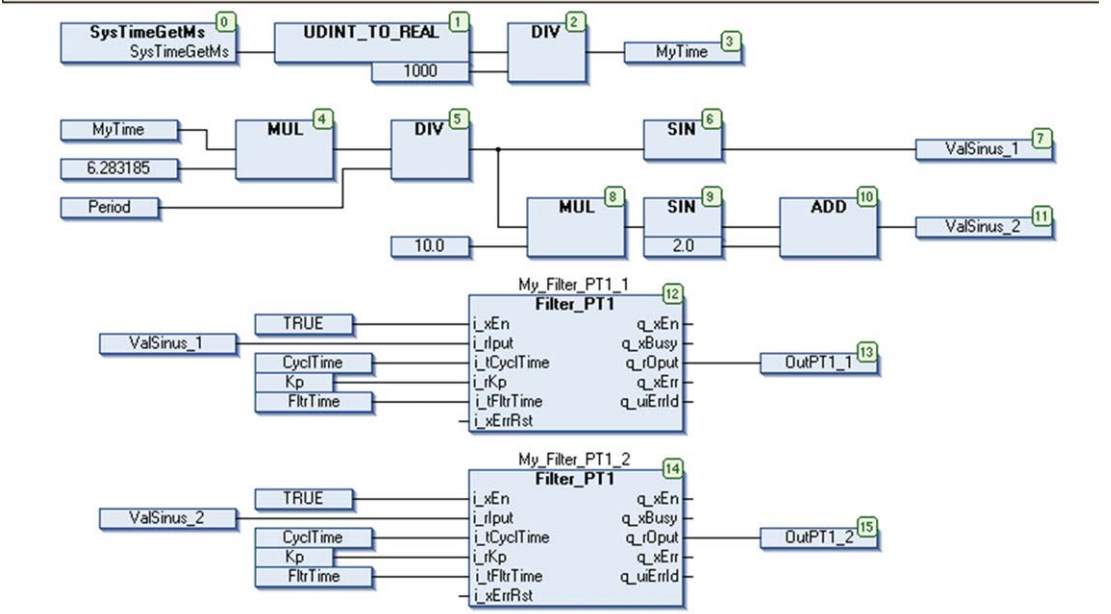
Instantiation and Usage Example

Example with a Frequency Signal

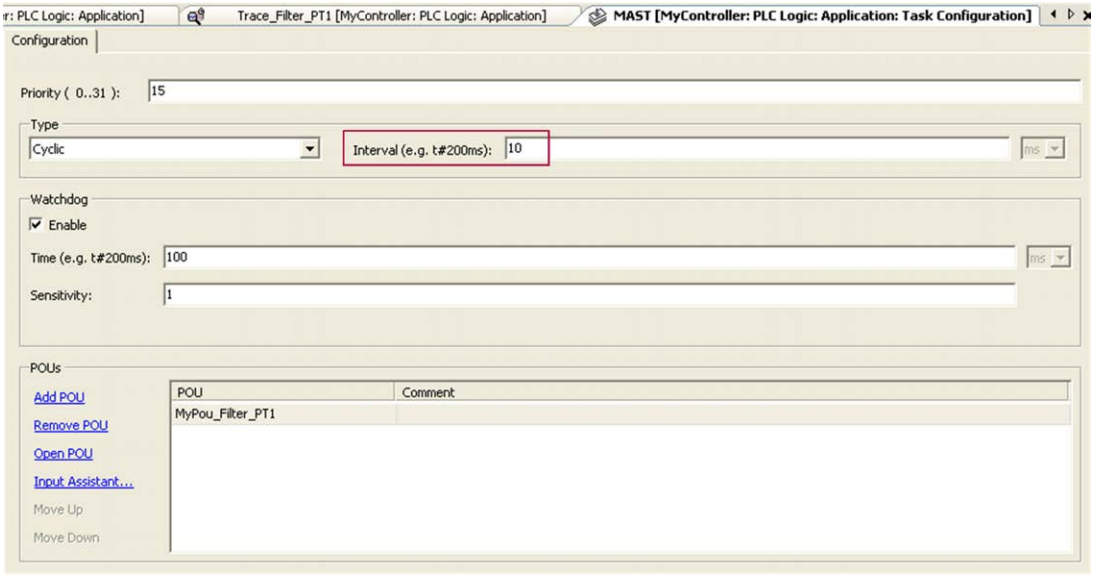
The program creates a Sinusoidal signal at a certain period (5 seconds/0.2 Hz) and a Sinusoidal signal one decade high (0.5 seconds/2 Hz).

MyPou_Filter_PT1 [MyController: PLC Logic: Application] | Trace_Filter_PT1 [MyController: PLC Logic: Application]

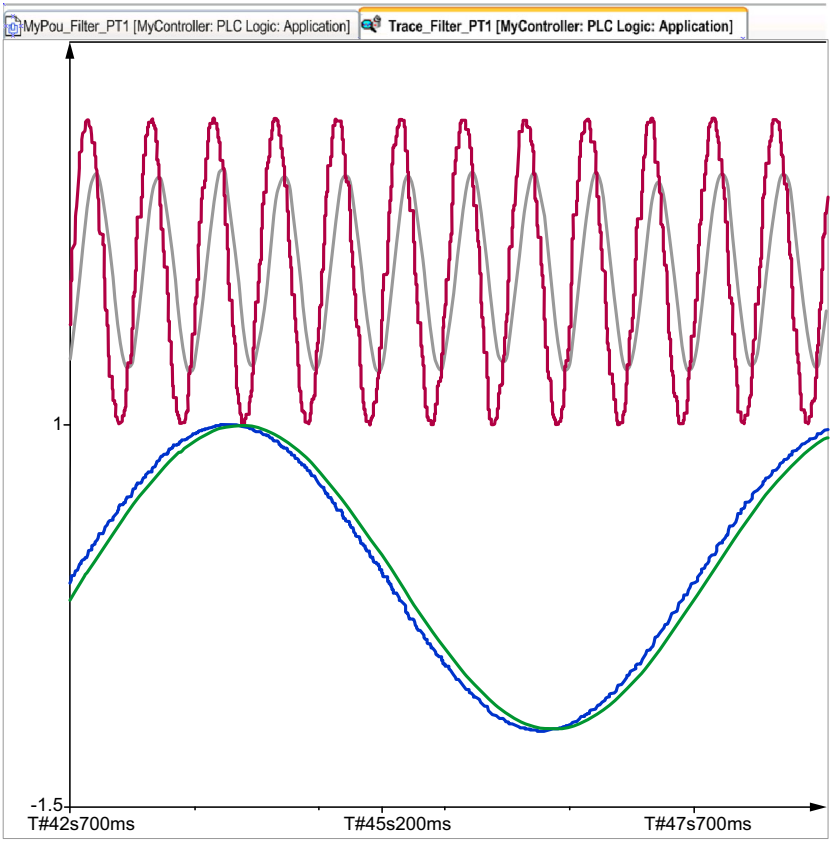
```
1  PROGRAM MyPou_Filter_PT1
2  VAR
3      FlagDemarrer    : BOOL;
4      Period          : REAL := 5.0;
5      ValSinus_1      : REAL;
6      ValSinus_2      : REAL;
7      MyTime          : REAL;
8      My_Filter_PT1_1 : Filter_PT1;
9      My_Filter_PT1_2 : Filter_PT1;
10     CyclTime        : TIME := T#10MS;
11     FltrTime        : TIME := T#100MS;
12     Kp              : REAL := 1.0;
13     OutPT1_1        : REAL;
14     OutPT1_2        : REAL;
15 END_VAR
```



The input `i_tCyclTime` of the `Filter_PT1` function block must have exactly the same value as the period of the POU in the MAST, here 10 milliseconds (See the red bordered area).

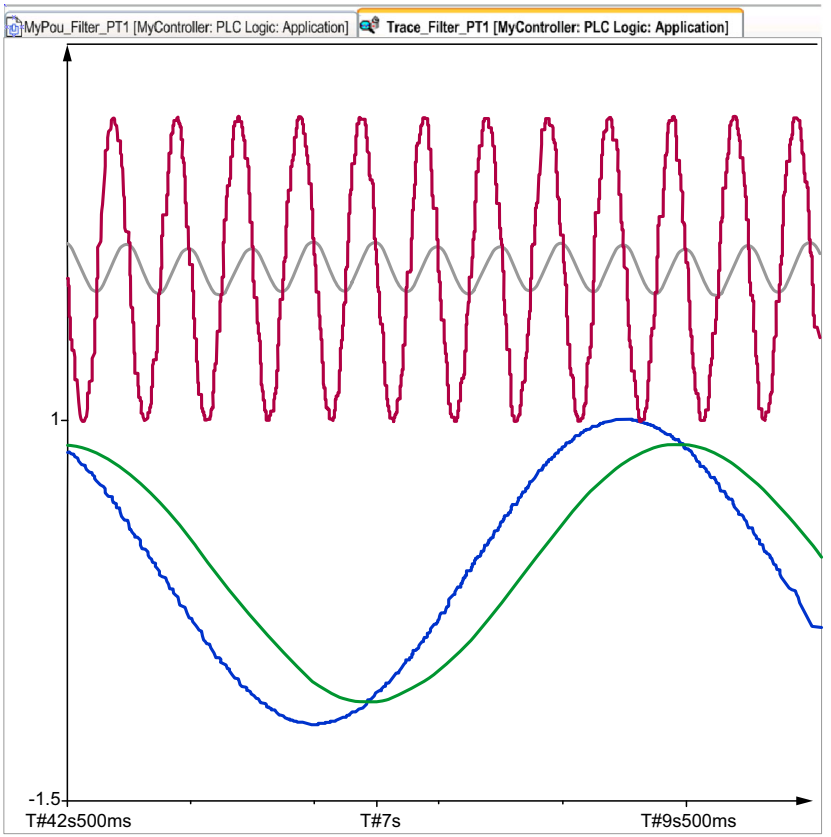


The result of the previous POU when the input `i_tFltrTime` equals 100 ms:



Blue `i_rIput` sinusoidal signal at 0.5 Hz (function block `My_Filter_PT1_1`)
Green `q_rOput` filtered signal (function block `My_Filter_PT1_1`)
Red `i_rIput` sinusoidal signal at 5 Hz (function block `My_Filter_PT1_2`)
Gray `q_rOput` filtered signal (function block `My_Filter_PT1_2`)

The result of the previous POU when the input `i_tFltrTime` equals 500 ms:



Blue `i_rIput` sinusoidal signal at 0.5 Hz (function block `My_Filter_PT1_1`)
Green `q_rOput` filtered signal (function block `My_Filter_PT1_1`)
Red `i_rIput` sinusoidal signal at 5 Hz (function block `My_Filter_PT1_2`)
Gray `q_rOput` filtered signal (function block `My_Filter_PT1_2`)

Part VI
Flip-Flops

Overview

This part describes the Flip-Flops family.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
24	JK_FlipFlop: Resetting/Setting Input to Flip Flop Output	165
25	JK_FlipFlop_MasterSlave: Resetting/Setting Input to Flip Flop Output	167
26	RS_FlipFlop: Resetting/Setting of Flip Flop Input/Output	171
27	SR_FlipFlop: Resetting/Setting of Flip Flop Input/Output	173
28	Toggle_FlipFlop: Toggling of Flip Flop Input/Output	175

Chapter 24
JK_FlipFlop: Resetting/Setting Input to Flip Flop Output

JK_FlipFlop Function Block

Pin Diagram

This figure shows the pin diagram of the JK_FlipFlop function block:



Functional Description

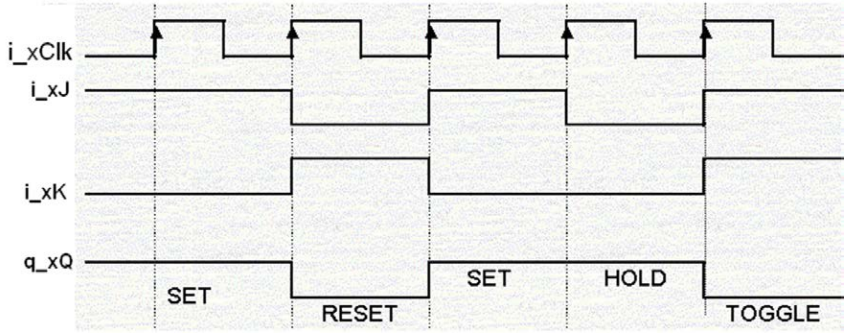
The JK_FlipFlop function block implements the truth table for JK flip-flop.

This function block refers to a flip-flop that obeys this truth table:

i_xClk	i_xJ	i_xK	q_xQ(n)	q_xQ(n+1)	Operation
0	X	X	X	Q(n)	Hold
RE	0	0	0	0	Hold
RE	0	0	1	1	Hold
RE	0	1	0	0	Reset
RE	0	1	1	0	Reset
RE	1	0	0	1	Set
RE	1	0	1	1	Set
RE	1	1	0	1	Toggle
RE	1	1	1	0	Toggle
n 'n' is the present state and (n+1) is the next state. **RE** Rising Edge					

The Reset input (i_xRst) resets the flip flop output q_xQ, whereas Set input (i_xSet) sets the flip flop output q_xQ.

Truth table represented as a time diagram:



Input Pin Description

This table describes the input pins of the `JK_FlipFlop` function block:

Input	Data Type	Description
`i_xJ`	`BOOL`	TRUE: `i_xJ` input active. FALSE: Disabled (factory setting)
`i_xK`	`BOOL`	TRUE: `i_xK` input active. FALSE: Disabled (factory setting)
`i_xClk`	`BOOL`	TRUE: Clock signal active. FALSE: Disabled (factory setting)
`i_xSet`	`BOOL`	TRUE: Sets the flip-flop output. FALSE: Disabled (factory setting)
`i_xRst`	`BOOL`	TRUE: Resets the flip-flop output. FALSE: Disabled (factory setting)

Output Pin Description

This table describes the output pins of the `JK_FlipFlop` function block:

Output	Data Type	Description
`q_xQ`	`BOOL`	Flip-flop output

Limitations

In JK flip-flop the inputs `i_xSet` and `i_xRst` have higher priority than `i_xJ` and `i_xK` inputs. When both the inputs `i_xSet` and `i_xRst` are either FALSE /TRUE then the output of the FB `q_xQ`, depends upon the inputs `i_xJ` and `i_xK` and `i_xClk`.

Chapter 25
JK_FlipFlop_MasterSlave: Resetting/Setting Input to Flip Flop Output

JK_FlipFlop_MasterSlave Function Block

Pin Diagram

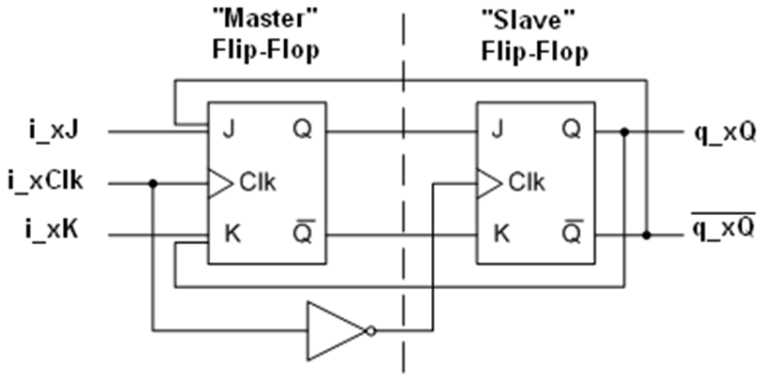
This figure shows the pin diagram of the JK_FlipFlop_MasterSlave function block:



Functional Description

The JK_FlipFlop_MasterSlave function block implements the truth table for master slave JK flip-flop. The master ouput is captured at rising edge of clock signal and output of slave is updated at falling edge of clock signal.

This diagram represents the internal construction of the JK_FlipFlop_MasterSlave function block.



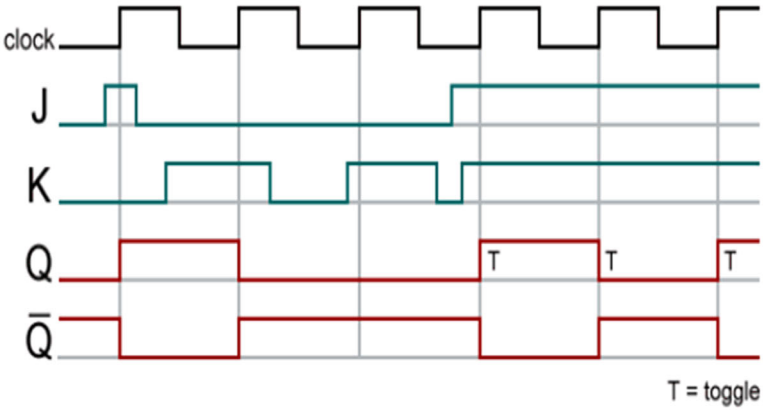
NOTE: The complementary output \q_xQ is not an output of the FB.

The `JK_FlipFlop_MasterSlave` refers to a flip-flop that obeys this truth table:

Set	Reset	CLK	J	K	Q
1	0	X	X	X	1
0	1	X	X	X	0
1	1	X	X	X	1^*
0	0	↑	0	0	Unv.
0	0	↑	1	0	1
0	0	↑	0	1	0
0	0	↑	1	1	Toggle
0	0	0	X	X	Unv.

The Reset input (`i_xRst`) resets the flip flop output `q_xQ`, whereas Set input (`i_xSet`) sets the flip flop output `q_xQ`.

Truth table represented as a time diagram:



Input Pin Description

This table describes the input pins of the JK_FlipFlop_MasterSlave function block:

Input	Data Type	Description
i_xJ	BOOL	TRUE: i_xJ input active. FALSE: Disabled (factory setting)
i_xK	BOOL	TRUE: i_xK input active. FALSE: Disabled (factory setting)
i_xClk	BOOL	TRUE: Clock signal active. FALSE: Disabled (factory setting)
i_xSet	BOOL	TRUE: Sets the flip-flop output. FALSE: Disabled (factory setting)
i_xRst	BOOL	TRUE: Resets the flip-flop output. FALSE: Disabled (factory setting)

Output Pin Description

This table describes the output pins of the JK_FlipFlop_MasterSlave function block:

Output	Data Type	Description
q_xQ	BOOL	Flip-flop output (True / False)

Limitations

In JK Master Slave flip-flop the inputs i_xSet and i_xRst have higher priority than i_xJ and i_xK inputs. When both the inputs i_xSet and i_xRst are either FALSE /TRUE then the output of the FB q_xQ depends upon the inputs i_xJ and i_xK and i_xClk.

Chapter 26
RS_FlipFlop: Resetting/Setting of Flip Flop Input/Output

RS_FlipFlop Function Block

Pin Diagram

This figure shows the pin diagram of the RS_FlipFlop function block:



Functional Description

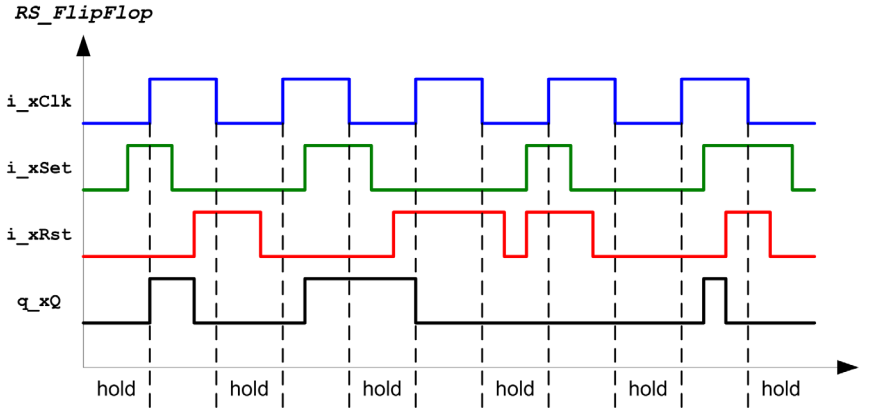
The RS_FlipFlop function block implements the truth table for RS flip-flop with reset priority.

The RS_FlipFlop refers to a flip-flop that obeys this truth table:

i_xClk	i_xSet	i_xRst	q_xQ(n+1)
0	X	X	Q(n)
1	0	0	Q(n)
1	0	1	0
1	1	0	1
1	1	1	0
n	'n' is the present state and (n+1) is the next state.		

It has two inputs namely, a Set input or i_xSet, and a Reset input or i_xRst. It also has one output q_xQ. When both set and reset inputs are high, priority is given to Reset input (i_xSet=1 and i_xRst=1).

Truth table represented as a time diagram:



Input Pin Description

This table describes the input pins of the RS_FlipFlop function block:

Input	Data Type	Description
i_xClk	BOOL	TRUE: Clock signal active. FALSE: Disabled (factory setting)
i_xSet	BOOL	TRUE: Sets the flip-flop output. FALSE: Disabled (factory setting)
i_xRst	BOOL	TRUE: Resets the flip-flop output. FALSE: Disabled (factory setting)

Output Pin Description

This table describes the output pins of the RS_FlipFlop function block:

Output	Data Type	Description
q_xQ	BOOL	Flip-flop output

Chapter 27
SR_FlipFlop: Resetting/Setting of Flip Flop Input/Output

SR_FlipFlop Function Block

Pin Diagram

This figure shows the pin diagram of the SR_FlipFlop function block:



Functional Description

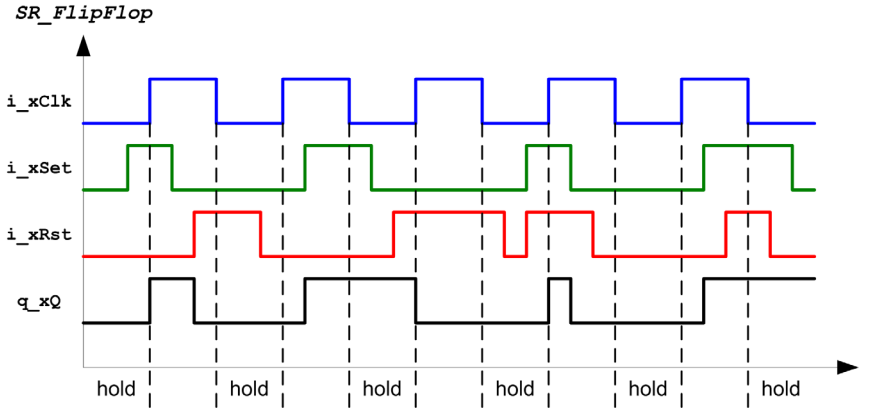
The SR_FlipFlop function block implements the truth table for SR flip-flop with set priority.

The SR_FlipFlop refers to a flip-flop that obeys this truth table:

i_xClk	i_xSet	i_xRst	q_xQ(n+1)
0	X	X	Q(n)
1	0	0	Q(n)
1	0	1	0
1	1	0	1
1	1	1	1
n 'n' is the present state and (n+1) is the next state.			

It has two inputs, namely, a Set input, or i_xSet, and a Reset input, or i_xRst. It also has one output, q_xQ. When both set and reset inputs are high, priority is given to Set input (i_xSet=1 and i_xRst=1).

Truth table represented as a time diagram:



Input Pin Description

This table describes the input pins of the SR_FlipFlop function block:

Input	Data Type	Description
i_xClk	BOOL	TRUE: Clock signal active. FALSE: Disabled (factory setting)
i_xSet	BOOL	TRUE: Sets the flip-flop output. FALSE: Disabled (factory setting)
i_xRst	BOOL	TRUE: Resets the flip-flop output. FALSE: Disabled (factory setting)

Output Pin Description

This table describes the output pins of the SR_FlipFlop function block:

Output	Data Type	Description
q_xQ	BOOL	Flip-flop output

Chapter 28
Toggle_FlipFlop: Toggling of Flip Flop Input/Output

`Toggle_FlipFlop` Function Block

Pin Diagram

This figure shows the pin diagram of the `Toggle_FlipFlop` function block:



Functional Description

The `Toggle_FlipFlop` function block implements the truth table for T(Toggle) flip-flop with set priority.

The `Toggle_FlipFlop` is a type of Flip-Flop which obeys this truth table:

i_xRst	**$i\_xT_{(n-1)}$**	**$i\_xT_{(n)}$**	**$q\_xQ_{(n)}$**	**$q\_xQ_{(n+1)}$**
0	0	0	X	$Q_{(n)}$
0	0	1	0	1
0	0	1	1	0
0	1	x	x	$Q_{(n)}$
1	x	x	x	0
n	‘n’ is the present state and (n+1) is the next state.			

It has two inputs, namely, `i_xT` input, and a Reset input, or `i_xRst`. It also has an output `q_xQ`.

Input Pin Description

This table describes the input pins of the `Toggle_FlipFlop` function block:

Input	Data Type	Description
`i_xT`	`BOOL`	Rising edge 0...1 toggles the flip-flop. Factory setting: FALSE
`i_xRst`	`BOOL`	TRUE: Resets the flip-flop output. FALSE: Disabled (factory setting)

Output Pin Description

This table describes the output pins of the `Toggle_FlipFlop` function block:

Output	Data Type	Description
`q_xQ`	`BOOL`	Flip-flop output

Part VII
Mathematical Functions

Overview

This part describes the Mathematical function family.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
29	Analysis: Calculating Integral and Derivative Values	179
30	Frequency_Multiplier: Implementing 32 Blinkers	181
31	Frequency_Output: Implementing a Frequency	187
32	Normalizer_With_Limiter: Scaling the Minimum and Maximum Input	195
33	ONE_SEC_PULSE: Providing Pulses Every One Second	199
34	Quantizer: Digitalizing the Input Value for the Interval	201
35	Signal_Saturation: Limiting to Upper and Lower Saturation Limit	203
36	Signal_Statistics: Calculating Maximum, Minimum, Average and Variance	209
37	Check_Divisor: Checking for Zero Division Condition	213

Chapter 29
Analysis: Calculating Integral and Derivative Values

`Analysis` Function Block

Pin Diagram

This figure shows the pin diagram of the `Analysis` function block:



Functional Description

The `Analysis` function block calculates the integral and derivative values of a series of input. The output starts with zero at the rising edge of `i_xEn`. The integral value increases in multiples of interval input.

In each scan both the integral output and the derivative output is updated based on the interval value.

An error is detected if the interval value is equal to/less than zero or if the input is out of range or if the integral or derivative outputs exceed $3.4e^{+38}$.

Integral = Integral + (Current Input + Previous Input) / 2 * Interval.

Derivative = (Current Input - Previous Input)/ Interval.

Example

Input = 10 (Previous input: 0), Interval = 10, then the outputs after the first cycle of execution are as below:

- Integral = 0 + (10+0) / 2 * 10 = 50
- Derivative = (10-0)/ 10 = 1

Input Pin Description

This table describes the input pins of the `Analysis` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: FB enabled FALSE: FB disabled
`i_rIput`	`REAL`	Input value Range: $1.17e^{-38}$...$3.4e^{38}$
`i_rItvl`	`REAL`	Input value Range: $1.17e^{-38}$...$3.4e^{38}$
`i_xErrRst`	`BOOL`	TRUE: Reset the detected error. (On rising edge) (Optional)

Output Pin Description

This table describes the output pins:

output	Data Type	Description
`q_xActv`	`BOOL`	function block status output
`q_rItgr`	`REAL`	Integral value Range: $1.17e^{-38}$...$3.4e^{38}$
`q_rDrvt`	`REAL`	Derivative value Range: $1.17e^{-38}$...$3.4e^{38}$
`q_xErr`	`BOOL`	TRUE: `i_rItvl` input <= 0 or `i_rIput` < $1.17e^{-38}$ or `i_rIput` > $3.4e^{+38}$ or `q_rItgr` > $3.4e^{+38}$ or `q_rDrvt` > $3.4e^{+38}$ FALSE: No detected error

Chapter 30
Frequency_Multiplier: Implementing 32 Blinkers

Overview

This chapter describes the `Frequency_Multiplier` function block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
`Frequency_Multiplier` Function Block	182
Without Hold Condition Description	184
Functionality With Condition Description	185

Frequency_Multiplier Function Block

Pin Diagram

This figure shows the pin diagram of the Frequency_Multiplier function block:



Functional Description

The Frequency_Multiplier function block implements 32 blinkers represented by the bits of output.

On every rising edge of enable signal, the blinker output starts with zero. The lowest bit changes its state after a time base period. The second bit blinks with half the frequency of the initial one. The third bit blinks with half the frequency of the second one and so on, until enable signal is reset. If i_xHold input is set, then current state of blinkers is Hold. If blinkers of type BOOL are required the function block DWORD_AS_BIT (Util Library) can be used.

The output is reset on the rising edge of Enable input.

Example

Frequencies (Enable = TRUE, Timebase = t#100ms, Hold = FALSE)

DWORD_AS_BIT (Input = Frequency ouput)

DWORD_AS_BIT.B00 is blinking with 100 ms

DWORD_AS_BIT.B01 is blinking with 200 ms

DWORD_AS_BIT.B02 is blinking with 400 ms

Input Pin Description

This table describes the input pins of the Frequency_Multiplier function block:

Input	Data Type	Description
i_xEn	BOOL	TRUE: FB enabled FALSE: Disabled
i_tBase	TIME	Time period Range: 1...4294967295 ms (≥ cycle time of controller)
i_xHold	BOOL	TRUE: Active FALSE: Disabled

Output Pin Description

This table describes the output pins of the `Frequency_Multiplier` function block:

output	Data Type	Description
`q_xActv`	`BOOL`	TRUE: FB enabled FALSE: Disabled
`q_dwOput`	`DWORD`	Output status Range: 0...4294967295

Without Hold Condition Description

Without Hold Condition

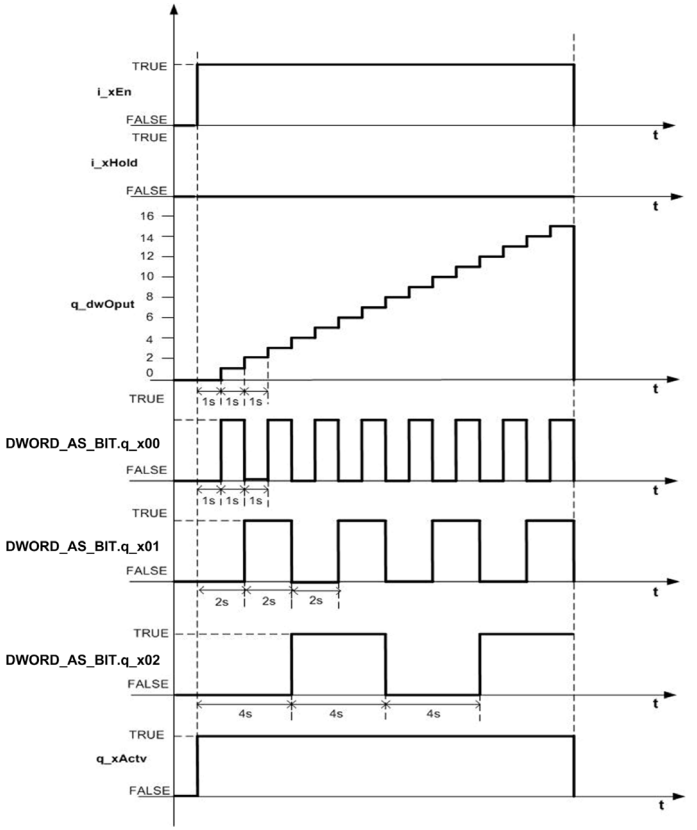
If input at pin `i_xEn` is high, `i_tBase` time is set to 1s and `i_xHold` input is FALSE, then `q_dwOput` is increased by one, after the completion of `i_tBase` time period.

`DWORD_AS_BIT` (Input: = `Frequency_Multiplier. q_dwOput`):

- `DWORD_AS_BIT.q_x00` is ON after 1 s of Enabling FB for Time base period
- `DWORD_AS_BIT.q_x01` is ON after 2 s of Enabling FB for 2 * Time base period
- `DWORD_AS_BIT.q_x02` is ON after 4 s of Enabling FB for 4 * Time base period

Timing Diagram

This figure shows the timing diagram for the `Frequency_Multiplier` function block without hold input:



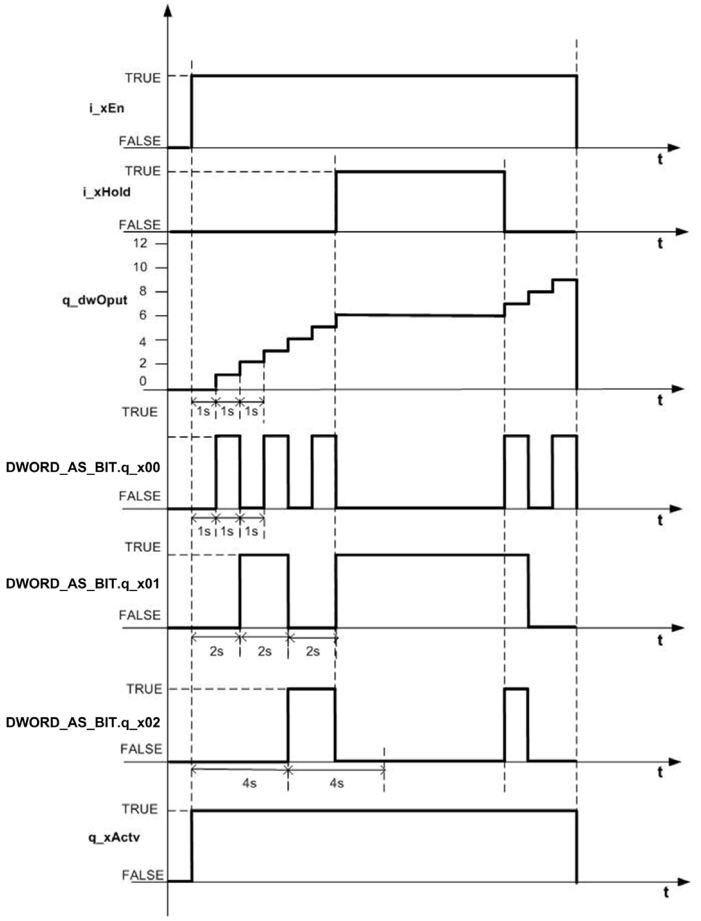
Functionality With Condition Description

Functionality Description

If input at pin `i_xEn` is high, `i_tBase` time is set to 1s and `i_xHold` input is TRUE, then `q_dwOput` HOLD the previous status of Blinkers. FB resumes with its normal operation again when `i_xHold` input goes to FALSE.

Timing Diagram

This figure show the timing diagram for the `Frequency_Multiplier` function block with hold input:

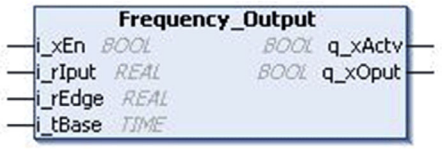


Chapter 31
Frequency_Output: Implementing a Frequency

`Frequency_Output` Function Block

Pin Diagram

This figure shows the pin diagram of the `Frequency_Output` function block:



Functional Description

The `Frequency_Output` function block implements frequencies.

A positive signal at `i_xEn` activates the block. The output signal starts with TRUE and holds this signal for a duration of (TimeBase * `i_rIput`) and then it changes to FALSE for a duration of TimeBase * (1-`i_rIput`).

If the input signal enters the edge range 0 to `i_rEdge`, the output signal no longer alternates and is continually FALSE. If it enters the range 1-`i_rEdge` to1 then the output signal is continuously TRUE.

If the signal at `i_xEn` is removed, the output signal remains at its current value until the function block is restarted through a positive signal at `i_xEn`.

Input Pin Description

This table describes the input pins of the `Frequency_Output` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: FB enabled FALSE: Disabled
`i_rIput`	`REAL`	Input value Range: 0.0...1.0
`i_rEdge`	`REAL`	Input value Range: 0.0...1.0
`i_tBase`	`TIME`	Time period Range: 0...4294967295 ms

Output Pin Description

This table describes the output pins of the `Frequency_Output` function block:

output	Data Type	Description
`q_xActv`	BOOL	function block active status
`q_xOput`	BOOL	Output value

Instantiation and Usage Example

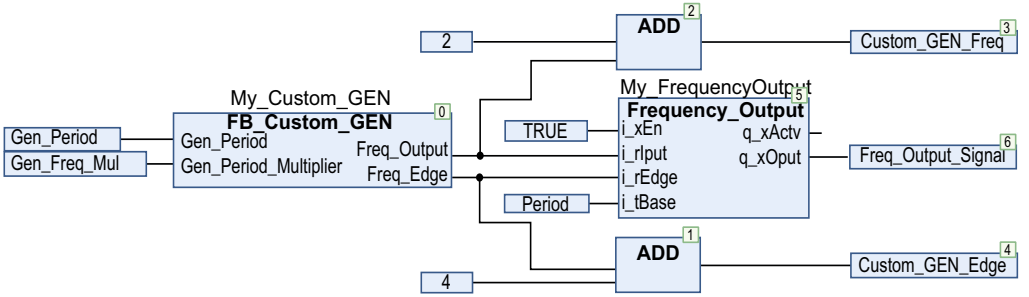
The example use the following custom signal generators that creates 2 signals :

- A 5 seconds period signal that will be used as the input for the `Frequency_Output` function block.
- A 50 seconds period signal that will be used as the edge for the `Frequency_Output` function block.

This POU has a period of 10 ms in the MAST.

MyPou_FrequencyOutput [MyController: PLC Logic: Application] | Trace_FrequencyOutput [MyController: PLC Logic: A...

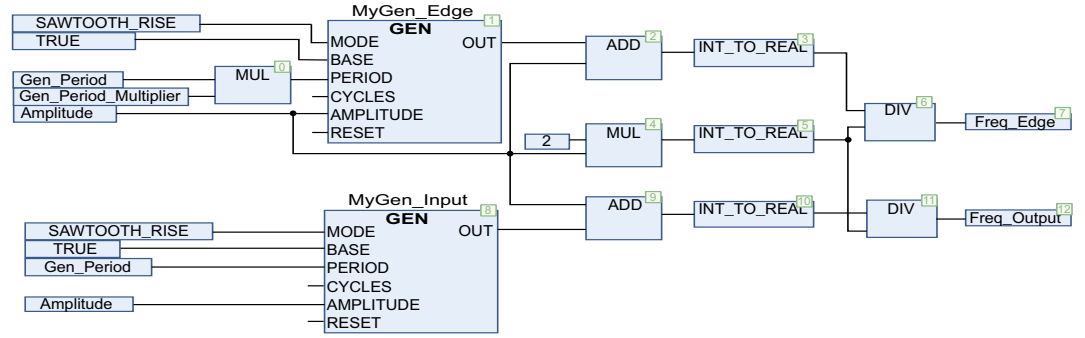
```
1  PROGRAM MyPou_FrequencyOutput
2  VAR
3   FlagDemarrer          : BOOL;
4
5   My_Custom_GEN         : FB_Custom_GEN;
6   Gen_Period            : TIME := T#5000MS;
7   Gen_Freq_Mul          : INT := 10;
8
9   My_FrequencyOutput    : Frequency_Output;
10  Period                : TIME := T#100MS;
11  Freq_Output_Signal    : BOOL;
12  Custom_GEN_Freq       : REAL;
13  Custom_GEN_Edge       : REAL;
14  Grid_Draw_1: REAL;
15 END_VAR
```



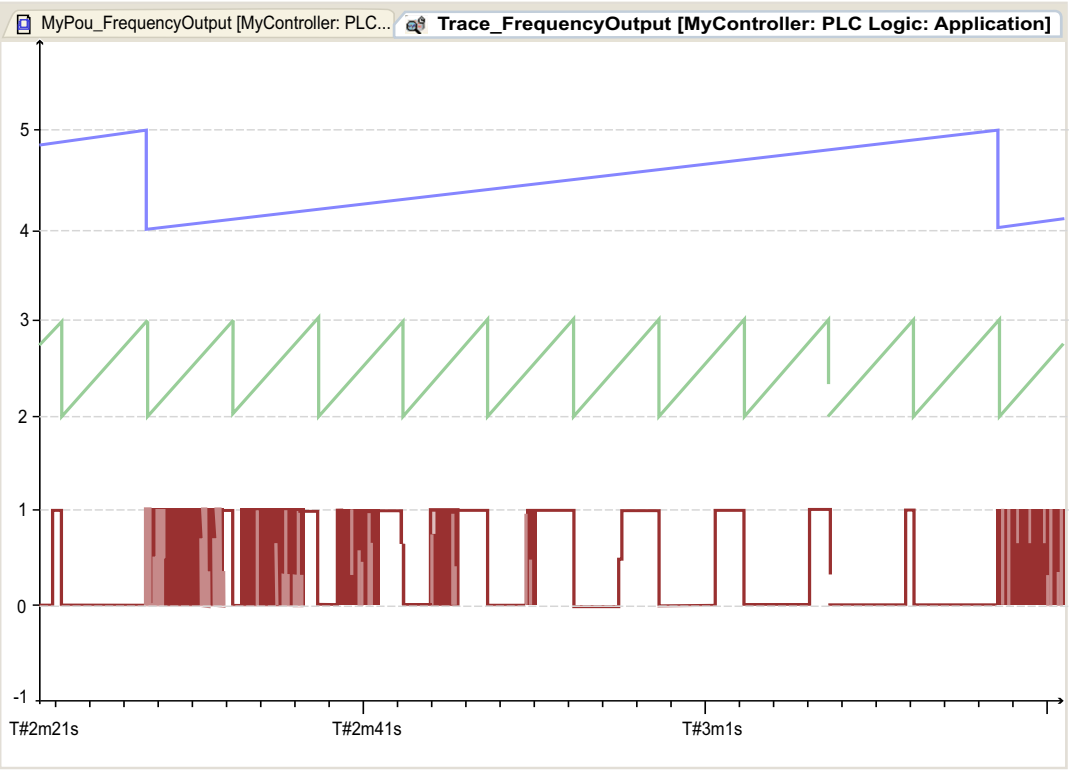
This figure shows the custom signal generator:

MyPou_FrequencyOutput.... | Trace_FrequencyOutput... | **FB_Custom_GEN [MyController: PLC Logic: Application]**

```
1  FUNCTION_BLOCK FB_Custom_GEN
2
3  VAR_INPUT
4      Gen_Period              : TIME;
5      Gen_Period_Multiplier   : INT;
6  END_VAR
7
8  VAR_OUTPUT
9      Freq_Output             : REAL;
10     Freq_Edge               : REAL;
11 END_VAR
12
13 VAR
14     MyGen_Edge              : GEN;
15     MyGen_Input             : GEN;
16     Amplitude               : INT:= 500;
17 END_VAR
18
```



This example gives the following results:

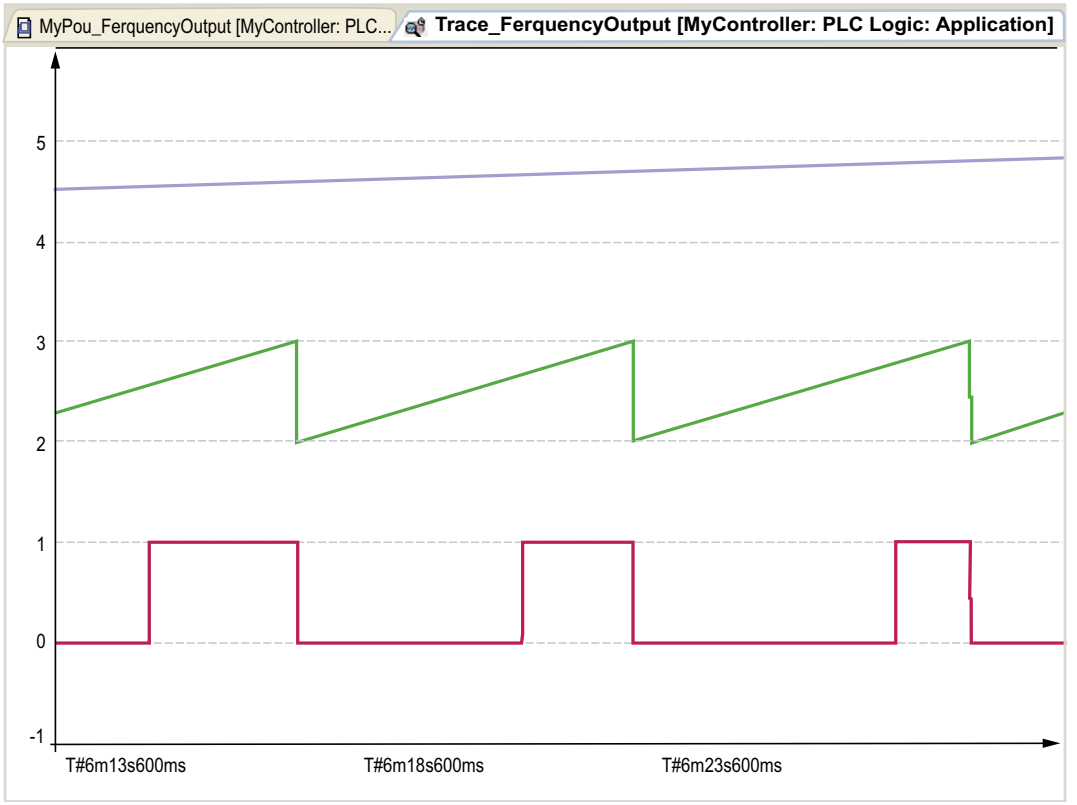


Blue `i_rEdge` signal, period of 50 seconds.
Green `i_rIput` signal, period of 5 seconds.
Red `q_xOput`, output of the `Frequency_Output` function block.

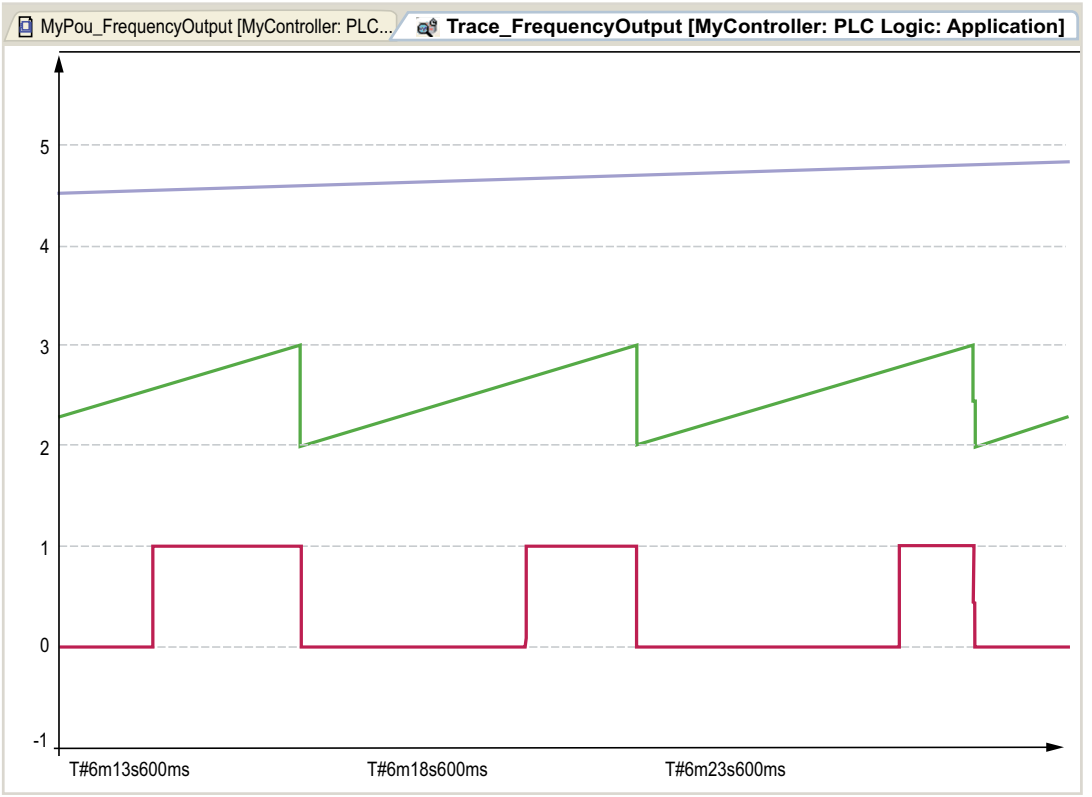
This table shows the truth table:

Edge level	Input level	Output
`i_rEdge` < 0.5	`i_rIput` < `i_rEdge`	FALSE
	`i_rEdge` < `i_rIput` < (1-`i_rEdge`)	PWM; duty = `i_rIput`
	(1 - `i_rEdge`) < `i_rIput`	TRUE
`i_rEdge` >= 0.5	`i_rIput` < `i_rEdge`	FALSE
	`i_rEdge` <= `i_rIput`	TRUE

Example with `i_rEdge` < 0.5:

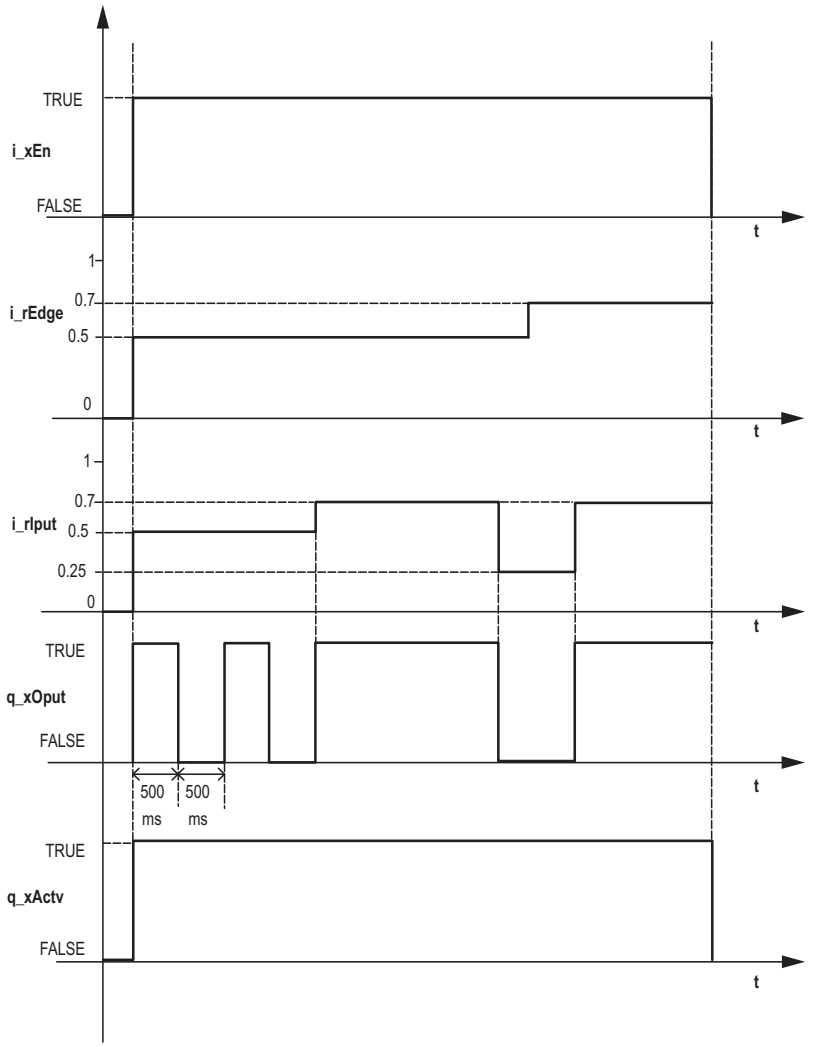


Example with `i_rEdge` >= 0.5:



Example: If the input at pin `i_xEn` is high, `i_rIput` input and initially `i_rEdge` input is set to 0.5 and `i_tBase` input is set to 1sec, then `q_xOput` is set for 500ms and reset for 500ms time period. After some time the input `i_rEdge` is changed to 0.7.

This figure shows the timing diagram of the above example:

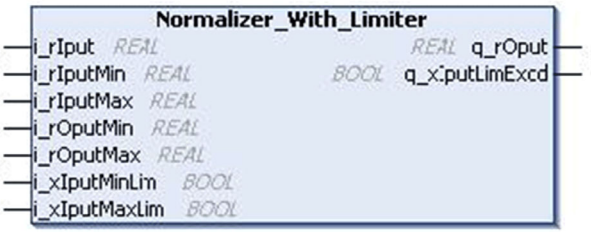


Chapter 32
Normalizer_With_Limiter: Scaling the Minimum and Maximum Input

`Normalizer_With_Limiter` Function Block

Pin Diagram

This figure shows the pin diagram of the `Normalizer_With_Limiter` function block:



Functional Description

The `Normalizer_With_Limiter` function block scales the minimum and maximum input value to the range of minimum and maximum output value. The input value can be limited to minimum and maximum output values.

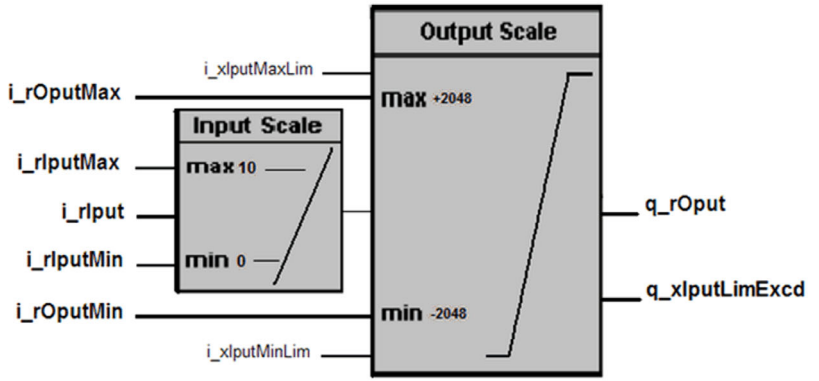
The output value can be limited to minimum and maximum output values using `i_xIputMinLim` and `i_xIputMaxLim`.

When the input exceeds the maximum/minimum input limit value `q_xIputLimExcd` will be TRUE.

Mathematical Backround

$$q\_rOput = \frac{(i\_rIput - i\_rIputMin) \times (i\_rOputMax - i\_rOputMin)}{(i\_rIputMax - i\_rIputMin)} + i\_rOputMin$$

Example



i_xIputMaxLim	i_xIputMinLim	i_rIput	q_rOput	q_xIputLimExcd
FALSE	FALSE	12.5	3072	TRUE
TRUE	FALSE	12.5	2048	TRUE
TRUE	FALSE	10	2048	FALSE
TRUE	FALSE	7.5	1024	FALSE
TRUE	TRUE	5	0	FALSE
FALSE	TRUE	2.5	-1024	FALSE
FALSE	TRUE	0	-2048	FALSE
FALSE	TRUE	-2.5	-2048	TRUE
FALSE	FALSE	-2.5	-3072	TRUE

Input Pin Description

This table describes the input pins of the `Normalizer_With_Limiter` function block:

Input	Data Type	Description
`i_rIput`	`REAL`	Input value Range: ±3.4e^{+38}
`i_rIputMin`	`REAL`	Minimum input Range: ±3.4e^{+38}
`i_rIputMax`	`REAL`	Maximum input value Range: ±3.4e^{+38}
`i_rOputMin`	`REAL`	Minimum output value Range: ±3.4e^{+38}
`i_rOputMax`	`REAL`	Maximum output value Range: ±3.4e^{+38}
`i_xIputMinLim`	`BOOL`	TRUE: Limit enabled for minimum input value FALSE:Limit disabled
`i_xIputMaxLim`	`BOOL`	TRUE: Limit enabled for maximum input value FALSE:Limit disabled

Output Pin Description

This table describes the output pins of the `Normalizer_With_Limiter` function block:

Output	Data Type	Description
`q_rOput`	`REAL`	Scale output value Range: ±3.4e^{+38}
`q_xIputLimExcd`	`BOOL`	Input value exceed limit value

Chapter 33
ONE_SEC_PULSE: Providing Pulses Every One Second

ONE_SEC_PULSE Function Block

Pin Diagram

This figure shows the pin diagram of the ONE_SEC_PULSE function block:



Functional Description

The ONE_SEC_PULSE function block generates pulses of one second duration at the output q_xOput.

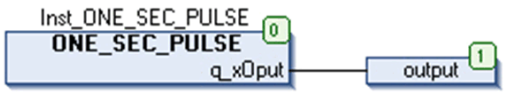
Output Pin Description

This table describes the output pins of the ONE_SEC_PULSE function block:

output	Data Type	Description
q_xOput	BOOL	Output Pulse Output state = TRUE during 1 cycle of task Frequency = 1 Hz

Instantiation and Usage Example

The above figure shows an instance of ONE_SEC_PULSE function block:



Chapter 34
Quantizer: Digitalizing the Input Value for the Interval

`Quantizer` Function Block

Pin Diagram

This figure shows the pin diagram of the `Quantizer` function block:



Functional Description

The `Quantizer` function block discretes the input value (–32768 to 32767) for a given interval.

If the input is more than the input range then the detected error output is TRUE and the output displays a zero value.

Mathematical Background

$$q\_rOput = i\_rItvl \times ROUND\left(\frac{i\_rIput}{i\_rItvl}; 0\right)$$

With Itvl = interval.

Example

Input	Interval	Output
32766.7	1	32767 Detected error = FALSE
32768	15	0 Detected error = TRUE
-32768	20	-32760 Detected error = FALSE
36.89	15	30 Detected error = FALSE

Input	Interval	Output
-47.98	-10	-50 Detected error = FALSE
-42.14	-10	-40 Detected error = FALSE
3456.78	80	3440 Detected error = FALSE
Between -4.99 to 4.99	10	0
Between 5 to 14.99	10	10

Input Pin Description

This table describes the input pins of the `Quantizer` function block:

Input	Data Type	Description
`i_rIput`	`REAL`	Input value Range: -32768...32767
`i_rItvl`	`REAL`	Quantization interval input value Range: $\pm 3.4e^{+38}$
`i_xErrRst`	`BOOL`	Reset the detected error. (On rising edge) (Optional)

Output Pin Description

This table describes the output pins of the `Quantizer` function block:

output	Data Type	Description
`q_rOput`	`REAL`	Output value Range: -32768...32767
`q_xErr`	`BOOL`	TRUE: Input limit exceed FALSE: No detected error

Chapter 35
Signal_Saturation: Limiting to Upper and Lower Saturation Limit

`Signal_Saturation` Function Block

Pin Diagram

This figure shows the pin diagram of the `Signal_Saturation` function block:



Functional Description

The `Signal_Saturation` function block limits the input signal to upper and lower saturation limit.

When low input value is more than the high value then the detected error output is TRUE and the output displays zero.

When the input exceeds the low/high input limit value, then `q_xIputLimExcd` is TRUE.

Input Pin Description

This table describes the input pins of the `Signal_Saturation` function block:

Input	Data Type	Description
`i_rIput`	`REAL`	Input value Range: $\pm 3.4e^{+38}$
`i_rLow`	`REAL`	Lower input value Range: $\pm 3.4e^{+38}$
`i_rHigh`	`REAL`	Higher input value Range: $\pm 3.4e^{+38}$
`i_xErrRst`	`BOOL`	Reset the detected error. (On rising edge) (Optional)

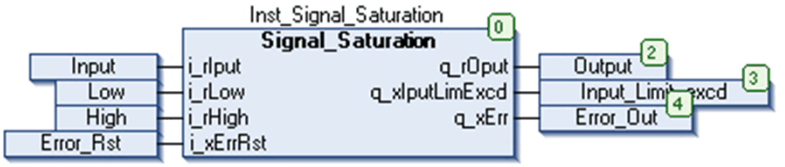
Output Pin Description

This table describes the output pins of the `Signal_Saturation` function block:

Output	Data Type	Description
`q_rOput`	`REAL`	Output value ±3.4e^{+38}
`q_xIputLimExcd`	`BOOL`	TRUE: Input value exceed limit value.
`q_xErr`	`BOOL`	TRUE: Input limit wrong FALSE: No detected error

Instantiation and Usage Example

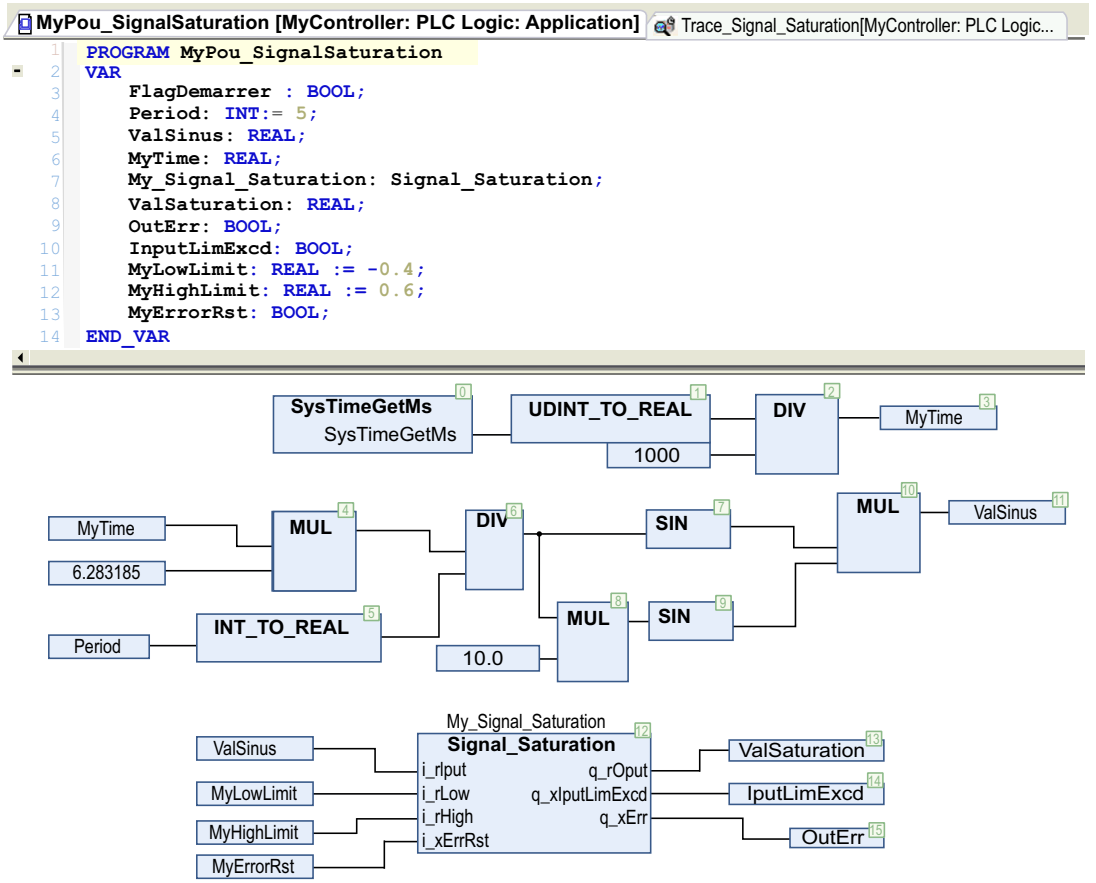
This figure shows an instance of the `Signal_Saturation` function block:



If input `i_rIput` is set to 4, `i_rLow` is set to 5 and `i_rHigh` is set to 10, then saturation output `i_rLow` value is 5 & `q_xIputLimExcd` is TRUE.

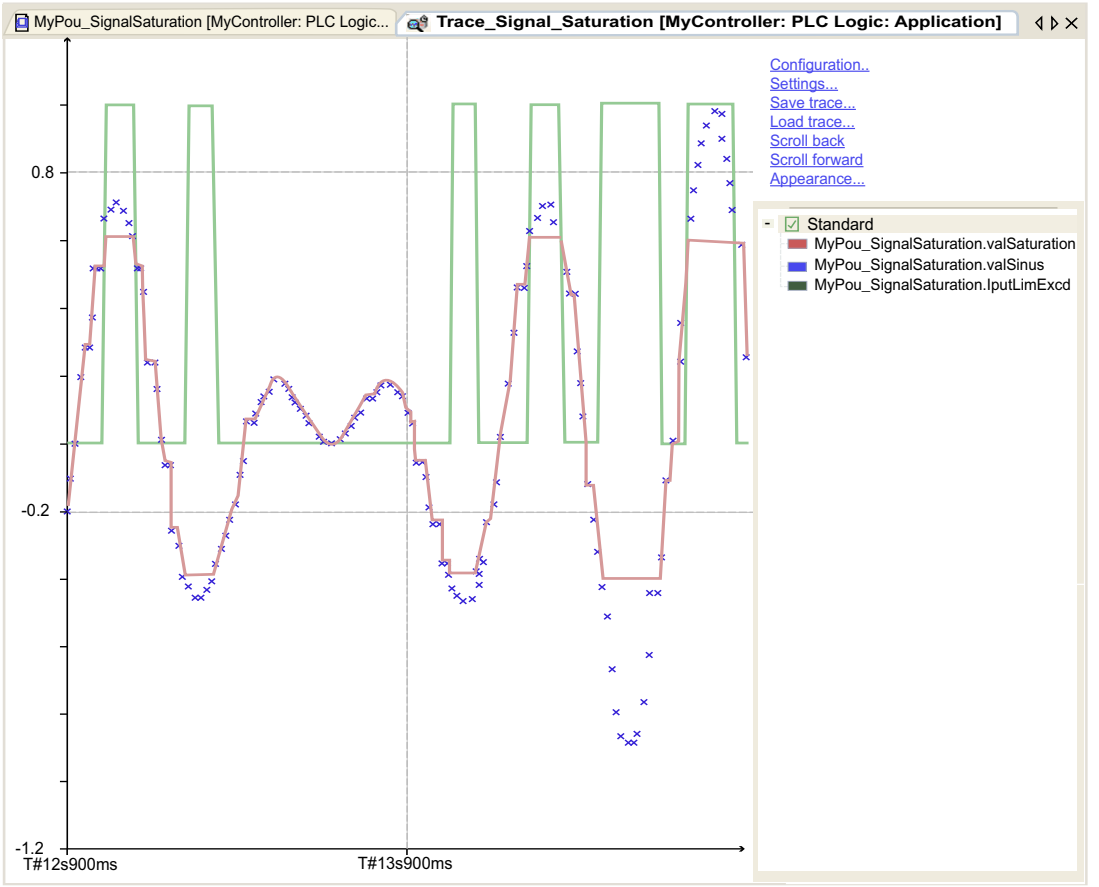
CFC Example

This figure shows the CFC example on `Signal_Saturation` implementation:



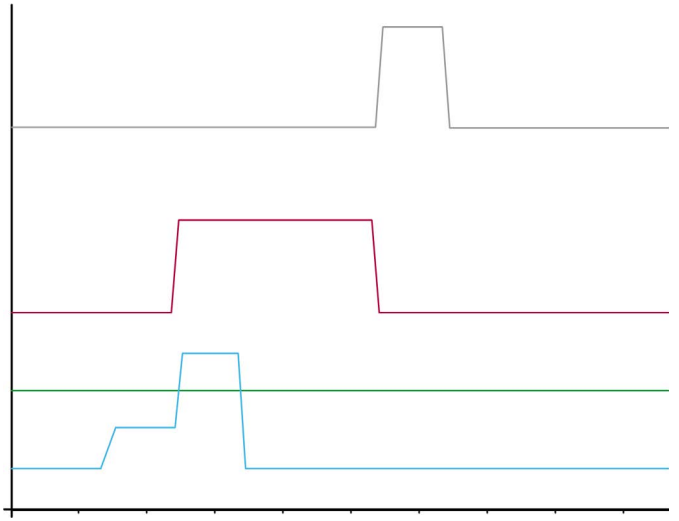
Timing Diagram

This figure shows the timing diagram for the `Signal_Saturation` function block:



Blue input signal
Red output signal, limited in the range [Low ; High]/[-0,4 ; +0,6].
Green `IputLimExcd`, TRUE when the input signal is out of the range.

This figure shows the timing diagram of `q_xErr` output:



Blue: `i_rLow` input signal
Green: `i_rHigh` input signal
Red: `q_xErr` output signal, TRUE as soon as `i_rLow` is higher than `i_rHigh`.
Gray: `i_xErrRst` input signal, reset the `q_xErr` output signal on rising edge while `i_rLow` is lower than `i_rHigh`.

Chapter 36
Signal_Statistics: Calculating Maximum, Minimum, Average and Variance

`Signal_Statistics` Function Block

Pin Diagram

This figure shows the pin diagram of the `Signal_Statistics` function block:



Functional Description

This function block calculates the maximum, minimum, average and variance of a series of input values.

This function block considers the input value in each controller scan cycle as a Sample.

Minimum Value

Minimum Output value is the value which is minimum amongst all the recorded samples.

Maximum value

Maximum Output value is the value which is maximum amongst all the recorded samples.

Average Value

The average value is equal to the sum of the observations (samples) divided by the number of observations (samples).

$$\bar{x} = \frac{1}{n}\left(\sum_{n} x_n\right)$$

Where:
- n = Number of samples recorded
- Xn = Input samples
- $\bar{x}$ = Calculated output

Variance Value

The variance is equal to the mean of the squares of samples minus the square of the mean (Average output value).

$$\bar{x} = \frac{1}{n}\left(\sum_{1}^{n}(x_n)^2\right) - \left(\frac{1}{n}\sum_{1}^{n}x_n\right)^2$$

Where:
- n = Number of samples recorded
- Xn = Input samples
- $\bar{x}$ = Calculated output

Example

- Statistics (Enable: = TRUE, Input: = 1, 2
- Minimum Output =1
- Maximum Output = 2
- Average = (1 + 2) / 2 = 1.5
- Variance = ((1 * 1 + 2 * 2) / 2) - (1.5 * 1.5) = 2.5 - 2.25 = 0.25

Input Pin Description

This table describes the input pins of the `Signal_Statistics` function block:

Input	Data Type	Description
`i_xEn`	`BOOL`	TRUE: FB enabled FALSE: FB Disabled
`i_rIput`	`REAL`	Bit position Range: $\pm 3.4e^{+38}$

Output Pin Description

This table describes the output pins of the `Signal_Statistics` function block:

output	Data Type	Description
`q_xActv`	BOOL	FB status output
`q_rMin`	REAL	Minimum value Range: $\pm 3.4e^{+38}$
`q_rMax`	REAL	Maximum value Range: $\pm 3.4e^{+38}$
`q_rAvge`	REAL	Average value Range: $\pm 3.4e^{+38}$
`q_rVrnc`	REAL	Variance value Range: $\pm 3.4e^{+38}$

Chapter 37
Check_Divisor: Checking for Zero Division Condition

`Check_Divisor` Function Block

Pin Diagram

This figure shows the pin diagram of the `Check_Divisor` function block:



Functional Description

The `Check_Divisor` function block checks for zero division condition.

If the divisor `i_rDvsr` is zero, then the output `q_rChkDiv` is 1, else if divisor not equal to 0, then output is equal to divisor.

Input Pin Description

This table describes the input pins of the `Check_Divisor` function block:

Input	Data Type	Description
`i_rDvsr`	`REAL`	Divisor output Range: ±3.4e^{+38}

Output Pin Description

This table describes the output pins of the `Check_Divisor` function block:

output	Data Type	Description
`q_rChkDiv`	`REAL`	Check division output Range: ±3.4e^{+38} **NOTE:** This output is not valid at the value 0.

Part VIII
Numerical Conversion Functions

Overview

This part describes the Numerical Conversion family.

What Is in This Part?

This part contains the following chapters:

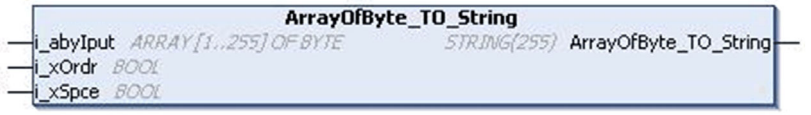
Chapter	Chapter Name	Page
38	ArrayOfByte_TO_String: Converting Array in Byte Format to String Format	217
39	DT_AS_WORD: Converting Date and Time as Word	221
40	DWORD_AS_WORD: Splitting the Double Word into Two Words	223
41	String_TO_ArrayOfByte: Output Array and ASCII Value of the Input String	225
42	WORD_AS_DWORD: Shifting Higher Word and Adding Lower Word	229

Chapter 38
ArrayOfByte_TO_String: Converting Array in Byte Format to String Format

`ArrayOfByte_TO_String` Function

Pin Diagram

This figure shows the pin diagram of the `ArrayOfByte_TO_String` function:



Functional Description

The output String [255] is the set of string characters which corresponds to the ASCII value of input array given in Byte format.

If Order input is TRUE, then the order of characters in the output string corresponds to the order of bytes at the input array. This means there is a 1:1 correspondence between order of input bytes and order of string characters returned at output as explained in example 1.

If the Order input is FALSE, then the order of characters in the output string will be such that, the string character corresponding to ASCII value at input[1] is displayed in output position[2] of output[1..255]. The string character corresponding to ASCII value at input[2] will be displayed in output position[1] of output[1..255].

Similarly the string character corresponding to ASCII value at input[3] will be displayed in output position[4] of output[1..255]. The string character corresponding to ASCII value at input[4] will be displayed in output position[3] of output[1..255] as explained in example 2.

Only if Order input is FALSE and Space input is TRUE and the number of input bytes at the input is odd, then a space character will be added prior to the last string character of the output string as shown in example 3 and 4.

But if the Order input is TRUE, then the Space input has no impact on the output as shown in example 6.

Example 1

Input: ARRAY [1..255] OF BYTE = 72, 69, 76, 76, 79;

Order: TRUE

Space: FALSE

String: 'HELLO'

As shown above, the order of characters in the output string corresponds to the order of input bytes, that is the byte value at the first position of Array[1..255] is 72 which corresponds to the string value at the first position of the output which is H. The byte value at second position of Array[1..255] is 69 which corresponds to string value at second position of the output which is E and so on.

Example 2

Input: ARRAY [1..255] OF BYTE = 65, 66, 67, 68, 69, 70, 71;

ByteOrder: FALSE

InsertSpace: FALSE

String: ‘BADCFEG’

As shown above, the order of characters in the output string is changed, that is the byte value at first position of Array [1...255] is 65 which corresponds to the string value at second position of the output which is A. The byte value at second position of Array[1..255] is 66 which corresponds to the string value at first position of the output which is B. Similarly the byte value at third position of Array[1..255] is 67 which corresponds to the string value at fourth position of the output which is C. The byte value at fourth position of Array[1..255] is 68 corresponds to the string value at third position of the output which is D and so on.

Example 3

Input: ARRAY [1..255] OF BYTE = 72, 69, 76, 76, 79;

Order: FALSE

Space: TRUE

String: ‘EHLL O’

Example 4

Input: ARRAY [1..255] OF BYTE = 65, 66, 67, 68, 69, 70, 71;

Order: FALSE

Space: TRUE

String: ‘BADCFE G’

As shown above in examples 3 & 4, the number of inputs is 5 in example 3 and 7 in example 4. Since 5 and 7 are odd numbers, the Order input is FALSE and Space input is TRUE. Hence the string outputs are 'EHLL O' AND 'BADCFE G' respectively.

NOTE: However if the number of bytes at the input are 255, Order input is FALSE and Space input is TRUE. The Space input becomes insignificant as explained in example 5 below.

Example 5

Input: ARRAY [1..250] OF BYTE = 65 and ARRAY [251..255] OF BYTE = 66, 67, 68, 69, 70;

Order: TRUE

Space: TRUE/FALSE

String[1..250]: 'A' and String[251..255] = ‘CBEDF’

As shown in the above example, if the number of bytes at the input is 255, the string output remains unaffected by the Space input

Example 6

Input: ARRAY [1..255] OF BYTE = 65, 66, 67, 68, 69, 70, 71;

Order: TRUE

Space: TRUE

String: ‘ABCDEFG’

As shown above, if the Order input is TRUE, the space input becomes insignificant.

Input Pin Description

This table describes the input pins of the `ArrayOfByte_TO_String` function:

Input	Data Type	Description
`i_abyIput`	`ARRAY [0..255] OF BYTE`	Input value Range: 0...255
`i_xOrdr`	`BOOL`	TRUE: In order of input FALSE: Swaps higher and lower byte
`i_xSpce`	`BOOL`	TRUE: Inserts space when `i_xOrdr` is low FALSE: No space is inserted.

NOTE: `I_xSpce` will insert a space character just before the last output string character when the Order input is FALSE and the number of input bytes is odd.

Output Pin Description

This table describes the output pins of the `ArrayOfByte_TO_String` function:

Output	Data Type	Description
`ArrayOfByte_TO_String`	`STRING(255)`	Output of string characters

NOTE: It is mandatory for the user to define the size of the string output as [255], else the size is taken as 80 by default.

Instantiation and Usage Example

This figure shows an instance of the `ArrayOfByte_TO_String` function:



With Order Input and Without Space Input

If input is:

- `i_abyIput` [255]
  - Input [1] = 65
  - Input [2] = 66
  - Input [3] = 67
  - Input [4] = 68
  - Input [5] = 69
- `i_xOrdr`: TRUE
- `i_xSpce`: FALSE

The `ArrayOfByte_TO_String` displays ‘ABCDE’.

With Order Input and With Space Input

If input is:

- `i_abyIput` [255]
  - Input [1] = 65
  - Input [2] = 66
  - Input [3] = 67
  - Input [4] = 68
  - Input [5] = 69
- `i_xOrdr`: FALSE
- `i_xSpce`: TRUE

The `ArrayOfByte_TO_String` displays ‘BADC E’

Chapter 39
DT_AS_WORD: Converting Date and Time as Word

DT_AS_WORD Function Block

Pin Diagram

This figure shows the pin diagram of the DT_AS_WORD function block:



Functional Description

The DT_AS_WORD function block extracts data from date and converts to equivalent words.

The DATE_AND_TIME input is converted to output in the form of WORD having year, month, date, hour, minute and second separately.

Example

With the input DT#2008-08-15-11:05:30, outputs are:

- Output year: 2008
- Output month: 8
- Output day: 15
- Output hours: 11
- Output minutes: 5
- Output seconds: 30

Input Pin Description

This table describes the input pins of the DT_AS_WORD function block:

Input	Data Type	Description
i_dtDttm	DT	Date and time input Range: 1970-01-01-00:00:00... 2106-02-07-06:28:15

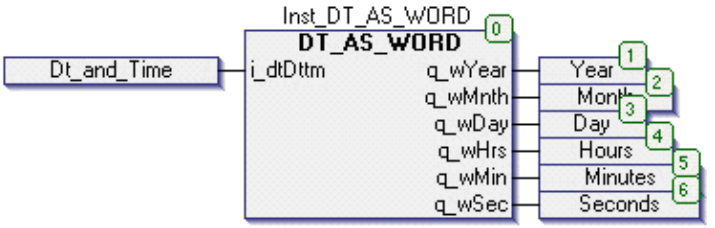
Output Pin Description

This table describes the output pins of the DT_AS_WORD function block:

Output	Data Type	Description
q_wYear	WORD	Year output Range: 1970...2106
q_wMnth	WORD	Month output Range: 1...12
q_wDay	WORD	Date output Range: 1...31
q_wHrs	WORD	Hours output Range: 1...23
q_wMin	WORD	Minutes output Range: 1...59
q_wSec	WORD	Seconds output Range: 1...59

Instantiation and Usage Example

This figure shows an instance of the DT_AS_WORD function block:



Operation of the function block is given in the below example:

- i_dtDttm: DT#2008-08-15-11:05:30
- q_wYear: 2008
- q_wMnth: 8
- q_wDay: 15
- q_wHrs: 11
- q_wMin: 5
- q_wSec: 30

Chapter 40
DWORD_AS_WORD: Splitting the Double Word into Two Words

DWORD_AS_WORD Function Block

Pin Diagram

This figure shows the pin diagram of the DWORD_AS_WORD function block:



Functional Description

The DWORD_AS_WORD function block converts an input value of data type DWORD to lower and higher outputs of type WORD.

The input double word i_dwIput is split into two words, higher q_wHigh and lower q_wLow words.

Input Pin Description

This table describes the input pins of the DWORD_AS_WORD function block:

Input	Data Type	Description
i_dwIput	DWORD	Input value Range: 0...4294967295

Output Pin Description

This table describes the output pins of the DWORD_AS_WORD function block:

Output	Data Type	Description
q_wLow	WORD	Lower word output value Range: 0...65535
q_wHigh	WORD	Higher word output value Range: 0...65535

Chapter 41
String_TO_ArrayOfByte: Output Array and ASCII Value of the Input String

`String_TO_ArrayOfByte` Function

Pin Diagram

This figure shows the pin diagram of the `String_TO_ArrayOfByte` function:



Functional Description

The `String_TO_ArrayOfByte` function is an output Array [255] of bytes is the ASCII value of the input String.

If the Order input is TRUE, then the order of the output values corresponds to the order of the string characters at the input. This means there is a 1:1 correspondence between the order of inputs and the order of ASCII value returned at the output as explained in example 1.

If the Order input is False, then the output is such that the ASCII value of string character at input[1] of input array[1..255] is displayed in position 2 of the output. The ASCII value of string character at input[2] of input array[1..255] is displayed in position 1 of the output. Similarly the ASCII value of string character at input[3] of input array[1..255] is displayed in position 4 of the output and the ASCII value of string character at input[4] of input array[1..255] is displayed in position 3 of the output as explained in example 2.

Example 1

If the Order input is TRUE then only output array is displayed in the order of string input as shown:

`i_sIput`='ABCDE'

`i_xOrdr`= TRUE

Then the string to array of byte output is:

- output [1] = 65
- output [2] = 66
- output [3] = 67
- output [4] = 68
- output [5] = 69
- output [6] = 0

As shown in the above example, Input [1] = A, its corresponding ASCII code is 65, displayed in output [1] position.

Similarly input [2] = B, its corresponding ASCII code is 66, displayed in output [2] position and so on.

Example 2

`i_sIput`='ABCDE'

`i_xOrdr`= FALSE

Then the string to array of byte output is:
- output [1] = 66
- output [2] = 65
- output [3] = 68
- output [4] = 67
- output [6] = 0
- output [5] = 69

As shown in the above example,

Input [1] = A, its corresponding ASCII code is 65, displayed in output [2] position.

Input[2] = B, and its corresponding ASCII code is 66, displayed in output [1] position.

Similarly input [3] = C, its corresponding ASCII code is 67, displayed in output [4] position.

Input [4] = D, its corresponding ASCII code is 68, displayed in output [3] position.

Similarly Input [5] = E, its corresponding ASCII code is 69, displayed in output [6] position.

Input [6] = (space), its corresponding ASCII code is " " (that is one blank space character) is displayed in output [5] position.

NOTE: However if the number of bytes at the input is 255, Order input is FALSE. Then the last ASCII value remains in the same position (refer example 3 below).

Input:
- `i_sIput` [1...250]='A'
- `i_sIput` [251...255]='BCDEF'

Order: FALSE

Output
- Output [1...250]:='65'
- Output [251...255]='CBEDF'

Input Pin Description

This table describes the input pins of the `String_TO_ArrayOfByte` function:

Input	Data Type	Description
`i_sIput`	`STRING [1...255]`	Input string value (1...255)
`i_xOrdr`	`BOOL`	TRUE: Output in order of input FALSE: Output swaps higher and lower byte.

NOTE: It is mandatory for the user to define the size of the string input[255], else the size is taken as 80 by default.

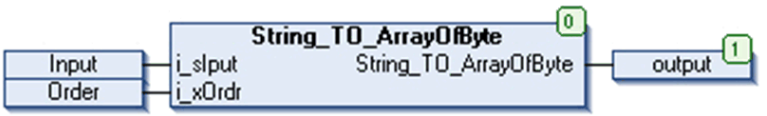
Output Pin Description

This table describes the output pins of the `String_TO_ArrayOfByte` function:

Output	Data Type	Description
String_TO_Array OfByte	`ARRAY [0...255] OF BYTE`	Array of ASCII values Range: 0...255

Instantiation and Usage Example

This figure shows an instance of the the `String_TO_ArrayOfByte` function:



With Order Input

`i_sIput` [255]:

- Input [1] = A
- Input [2] = B
- Input [3] = C
- Input [4] = D
- Input [5] = E

`i_xOrdr`: TRUE

The `String_TO_ArrayOfByte` displays '65, 66, 67, 68, 69'

Without Order Input

`i_sIput` [255]:

- Input [1] = A
- Input [2] = B
- Input [3] = C
- Input [4] = D
- Input [5] = E

`i_xOrdr`: FALSE

The `String_TO_ArrayOfByte` displays '66, 65, 68, 67, 69'

Chapter 42
WORD_AS_DWORD: Shifting Higher Word and Adding Lower Word

WORD_AS_DWORD Function Block

Pin Diagram

This figure shows the pin diagram of the WORD_AS_DWORD function block:



Functional Description

The WORD_AS_DWORD function block merges two input values of data type WORD into a single output of type DWORD.

The higher word input i_wHigh is shifted to the left by 4 nibbles and adds the lower word input i_wLow to obtain a Dword output q_dwOput.

Input Pin Description

This table describes the input pins of the WORD_AS_DWORD function block:

Input	Data Type	Description
i_wLow	WORD	Lower word input value Range: 0...65535
i_wHigh	WORD	Higher word input value Range: 0...65535

Output Pin Description

This table describes the output pins of the WORD_AS_DWORD function block:

Output	Data Type	Description
q_dwOput	DWORD	Output Dword value Range: 0...4294967295

Part IX
Physical Conversion

Overview

This part describes the Physical Conversion family.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
43	Celsius_TO_Fahrenheit: Converting Celsius to Fahrenheit	233
44	Celsius_TO_Kelvin: Converting Celsius to Kelvin	235
45	Fahrenheit_TO_Celsius: Converting Fahrenheit to Celsius	237
46	Frequency_TO_Period: Calculating Period of Time of Given Frequency	239
47	Kelvin_TO_Celsius: Converting Kelvin to Celsius	241
48	Period_TO_Frequency: Calculating Frequency of Given Time	243

Chapter 43
Celsius_TO_Fahrenheit: Converting Celsius to Fahrenheit

`Celsius_TO_Fahrenheit` Function

Pin Diagram

This figure shows the pin diagram of the `Celsius_TO_Fahrenheit` function:



Functional Description

The `Celsius_TO_Fahrenheit` function converts temperature in Celsius to Fahrenheit.

Use `Fahrenheit_TO_Celsius` for the reverse process.

Formula: T_Fahrenheit = [(T_Celcius * 1.8) + 32]

Input Pin Description

This table describes the input pins of the `Celsius_TO_Fahrenheit` function:

Input	Data Type	Description
`i_rIput`	REAL	Input value in Celsius Range: ±1.89e^{38} (higher values result INFINITY at the output).

Output Pin Description

This table describes the output pins of the `Celsius_TO_Fahrenheit` function:

Output	Data Type	Description
`Celsius_TO_Fahrenheit`	REAL	Output value in Fahrenheit

Chapter 44
Celsius_TO_Kelvin: Converting Celsius to Kelvin

Celsius_TO_Kelvin Function Block

Pin Diagram

This figure shows the pin diagram of the Celsius_TO_Kelvin function block:



Functional Description

The Celsius_TO_Kelvin function block converts the value of Celsius unit of REAL type to Kelvin unit. This result will be a REAL number.

The i_rIput pin is used to enter Celsius.

The q_rOput pin returns the equivalent Kelvin value in REAL data type.

Formula: Kelvin = Celsius + 273.15

Input Detected Error

The q_xErr pin becomes TRUE if invalid Celsius value is entered in pin i_rIput and the pin q_rOput returns to 0 as the temperature in Kelvin unit cannot be less than 0.

This detected error pin is reset on valid input entry.

Input Pin Description

This table describes the input pins of the Celsius_TO_Kelvin function block:

Input	Data Type	Description
i_rIput	REAL	Input value in Celsius Range: -273.15...3.4e^{+38}

Output Pin Description

This table describes the output pins of the `Celsius_TO_Kelvin` function block:

Output	Data Type	Description
`q_xErr`	`BOOL`	TRUE: Invalid input FALSE: Valid input
`q_rOput`	`REAL`	Output value in Kelvin Range: 0...$3.4e^{+38}$

The `i_rIput` input cannot be set to less than -273.15 because the equivalent Kelvin value is less than 0 which is theoretically not possible.

Chapter 45
Fahrenheit_TO_Celsius: Converting Fahrenheit to Celsius

`Fahrenheit_TO_Celsius` Function

Pin Diagram

This figure shows the pin diagram of the `Fahrenheit_TO_Celsius` function:



Functional Description

The `Fahrenheit_TO_Celsius` function converts temperature in Fahrenheit to Celsius.

Use `Celsius_TO_Fahrenheit` for the reverse process.

Formula: T_Celcius = [(T_Fahrenheit - 32) / 1.8]

Input Pin Description

This table describes the input pins of the `Fahrenheit_TO_Celsius` function:

Input	Data Type	Description
`i_rIput`	REAL	Input value in Fahrenheit Range: $\pm 3.4e^{+38}$

Output Pin Description

This table describes the output of the `Fahrenheit_TO_Celsius` function:

Output	Data Type	Description
`Fahrenheit_TO_Celcius`	REAL	Output value in Celsius Range: $\pm 1.89e^{+38}$

Chapter 46
Frequency_TO_Period: Calculating Period of Time of Given Frequency

Frequency_TO_Period Function

Pin Diagram

This figure shows the pin diagram of the Frequency_TO_Period function:



Functional Description

The Frequency_TO_Period function converts frequency (Hertz) value of type REAL to time. This result is of TIME type.

The period of time is calculated with the given frequency. The frequency is set in i_rIput pin in REAL data format. The equivalent time value is returned in Frequency_TO_Period pin in TIME data format.

Period = 1 / Frequency

NOTE: If the input is not in the previous range, the output is zero.

Input Pin Description

This table describes the input pins of the Frequency_TO_Period function:

Input	Data Type	Description
i_rIput	REAL	Input Frequency 0.0...1000.0Hz

Output Pin Description

This table describes the output pins of the Frequency_TO_Period function:

Output	Data Type	Description
Frequency_TO_Period	TIME	Period of time of the frequency input. 0...4294967295 ms

Chapter 47
Kelvin_TO_Celsius: Converting Kelvin to Celsius

Kelvin_TO_Celsius Function Block

Pin Diagram

This figure shows the pin diagram of the Kelvin_TO_Celsius function block:



Functional Description

The Kelvin_TO_Celsius function block converts the value in Kelvin unit of type REAL to Celsius unit. The result is a REAL number.

The pin i_rIput is used to enter Kelvin.

The pin q_rOput returns the equivalent Celsius value in REAL data type.

Formula: Celsius = Kelvin – 273.15

Input Detected Error

The q_xErr pin of type BOOL becomes TRUE if the invalid Kelvin value (ie < 0) is entered in the pin i_rIput and the pin q_rOput returns –273.15, because the equivalent Celsius value for the minimum Kelvin value is –273.15.

This detected error pin is reset on valid input entry.

Input Pin Description

This table describes the input pins of the Kelvin_TO_Celsius function block:

Input	Data Type	Description
i_rIput	REAL	Input value in Kelvin Range: 0...3.4e^{+38}

Output Pin Description

This table describes the output pins of the `Kelvin_TO_Celsius` function block:

Output	Data Type	Description
`q_xErr`	BOOL	TRUE: Invalid input. FALSE: Valid input.
`q_rOput`	REAL	Output value in Celsius Range: -273.15...$3.4e^{+38}$

Limitations

The `i_rIput` input cannot be set to less than 0 because, theoretically Kelvin value cannot be less than 0.

Chapter 48
Period_TO_Frequency: Calculating Frequency of Given Time

`Period_TO_Frequency` Function

Pin Diagram

This figure shows the pin diagram of the `Period_TO_Frequency` function:



Functional Description

The `Period_TO_Frequency` function converts time of type `TIME` to frequency(Hertz).This result is a `REAL` number.

This function calculates the frequency of given period of time. Time is set in `i_rIput` pin in `TIME` data format. The equivalent frequency value is returned in `Period_TO_Frequency` pin in `REAL` data format.

Frequency = 1 / Period

Input Pin Description

This table describes the input pins of the `Period_TO_Frequency` function:

Input	Data Type	Description
`i_tIput`	`TIME`	Input time value Range: 0...4294967295 ms

NOTE: If the input is not in the previous range, the output will be zero.

Output Pin Description

This table describes the output pins of the `Period_TO_Frequency` function:

Output	Data Type	Description
`Period_TO_Frequency`	`REAL`	Equivalent frequency of the time input Range: 0...1000 Hz

Part X
Utilities

Overview

This part describes the utilities function block family.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
49	Hour_Meter: Accumulating Operating Hours	247
50	Operation_Mode: Selecting Operating Mode	257

Chapter 49
Hour_Meter: Accumulating Operating Hours

Overview

This chapter describes the `Hour_Meter` function block.

What Is in This Chapter?

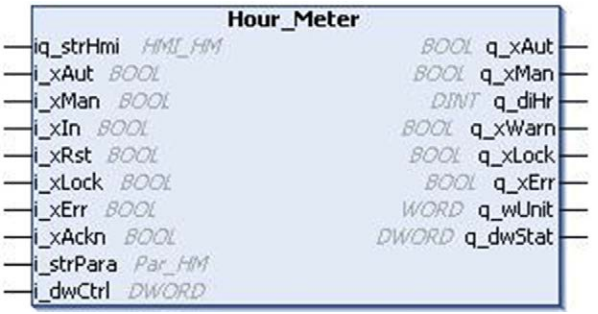
This chapter contains the following topics:

Topic	Page
`Hour_Meter` Function Block	248
Input Pin Description	250
Output Pin Description	251
Input – Output Pin	251
Structures Used	252
Control Word Bit Description	252
Status Word	253
Instantiation and Usage Example	254

Hour_Meter Function Block

Pin Diagram

This figure shows the pin diagram of the Hour_Meter function block:



Functional Description

The Hour_Meter function block is used for accumulating the operating hours of various devices.

Limitations

- If an error is detected, time is not calculated for the period till the detected error is acknowledged after which time calculation resumes from the value at the arrival of detected error. Hence, if the device for which UP time is being calculated is active during the error detected state, then the accumulated time is lesser than the actual time.
- For a wrong value of time unit specified in the variable i_strPara.wTypeTime, the function block displays the previous value of calculated time when the time unit input is correct, but the function block does not inform the user about this erroneous condition.
- Even if the block is locked by the external Interlock input, the function block can receive, generate and display a detected error.
- In Locked mode, the function block can reset the detected error by receiving the acknowledgement input.
- The calculated time value at the output is only in seconds or minutes or hours. The user should convert it into a format like HH:MM:SS if required.
- The structure variable i_strPara holds the warning time value and the time unit. The user must take special precaution not to accidentally change the time units as this can result in false alarm generation.

Operation Modes

The accumulation can take place either in manual mode or automatic mode:

- **Automatic Mode:** The automatic mode is selected through the input pin `i_xAut`. When the input `i_xIn` is set to TRUE the block accumulates the time and stops when the `i_xIn` is set to FALSE. The accumulated time is available at the output pin `q_diHr`.
- **Manual Mode:** The Manual mode is activated by the pin `i_xMan`. When the input `i_xIn` is set to TRUE the block accumulates the time and stops when the `i_xIn` is set to FALSE.The accumulated time is available at the output pin `q_diHr`. The accumulation can be inhibited by the command bit `i_dwCtrl`.

The block is de-activated on controller start and remains in the specified mode until a new one is selected. If both inputs are set to 1, then the operation mode is invalid.

Resetting Value

The reset of the output `q_diHr` is executed by a rising edge at the input `i_xRst` in automatic mode or by a command bit in manual mode.

The reset value of the output `q_diHr` is set to the value `i_strPara.diSp` (Set Point). Additionally the detected signal `q_xWarn` is set, when the output `q_diHr` exceeds the detected error limit specified by the parameter `i_strPara.diWaitTime`.

Setting Output Value Type

The parameter `i_strPara.wTypeTime` sets the unit of the output value. The selectable are seconds, minutes and hours. The accumulation function does not depend on this value, but is always done on the base of seconds.

Running Conditions

The counting takes place only, if the interlock input `i_xLock` is set to FALSE. An active interlock signal inhibits the operation of the hour meter. An active interlock is indicated at the output `q_xLock`.

The function block sets the detected error signal, if the detected error input `i_xErr` is set to TRUE (external detected error) or if the operation mode is invalid (internal detected errors). The detected errors are indicated in the HMI. To reset the detected error output the detected error has to be acknowledged by a rising edge on the input `i_xAckn` or by using bit 16 of the signal `i_dwCtrl`.

Input Pin Description

Input Pin Description

This table describes the input pins of the `Hour_Meter` function block:

Input	Data Type	Description
`i_xAut`	`BOOL`	TRUE: Auto mode enabled FALSE: Disabled (factory setting)
`i_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled (factory setting)
`i_xIn`	`BOOL`	TRUE: Block accumulates the time FALSE: Disabled (factory setting)
`i_xRst`	`BOOL`	TRUE: Sets the output back to the preset value entered at input `i_strPara`. FALSE: Disabled (factory setting) (Optional)
`i_xLock`	`BOOL`	Interlock input for operation TRUE: Interlock is active FALSE: No Interlock. (factory setting) (Optional)
`i_xErr`	`BOOL`	TRUE: External detected error is active FALSE: No external detected error (factory setting)
`i_xAckn`	`BOOL`	Acknowledgement is done with a rising edge. Input to acknowledge internal and external detected errors indicated at the output `q_xErr`.
`i_strPara`	`STRUCT Par_HM`	Structure with parameters for this block. Refer to the *(see page 252)*`Par_HM` description.
`i_dwCtrl`	`DWORD`	Command bits to interact from the HMI Range: 0...4294967295 Refer to the control word bit description *(see page 252)*.

Output Pin Description

Output Pin Description

This table describes the output pins of the `Hour_Meter` function block:

Output	Data Type	Description
`q_xAut`	`BOOL`	TRUE: Automatic mode enabled FALSE: Disabled
`q_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled
`q_diHr`	`DINT`	Accumulated operating time in seconds, minutes or hours. Range: -2147483648...2147483647
`q_xWarn`	`BOOL`	TRUE: Signal that the time at hour has exceeded a warn time value FALSE: Disabled.
`q_xLock`	`BOOL`	TRUE: Interlock is active. FALSE: No interlock Indicates that the operation is blocked by an interlock (input `i_xLock`)
`q_xErr`	`BOOL`	TRUE: detected error is active FALSE: No detected error
`q_wUnit`	`WORD`	Indicates the unit of the displayed operating time at the output hour: • 16#01: Seconds • 16#02: Minutes • 16#04: Hours
`q_dwStat`	`DWORD`	Status bits to be displayed in the HMI Range: 0...4294967295

Input – Output Pin

Input - Output from HMI

This table describes the Input – Output pin of the `Hour_Meter` function block:

Input - Output	Data Type	Description
`iq_strHmi`	`STRUCT HMI_HM`	Structure to interface with the HMI Refer to used structures *(see page 252)*.

Structures Used

Par_HM

Structure Element	Type	Description
diSp	DINT	Preset value for the hour meter time output used on reset
diWaitTime	DINT	Time value for activating the detected error signal
wTypeTime	WORD	Set the accumulated time at the output in seconds, minutes or hours: ● 16#01: Seconds ● 16#02: Minutes ● 16#04: Hours

HMI_HM

Structure Element	Type	Description
diVal	DINT	Accumulated operating time in seconds, minutes or hours.
diSp	DINT	Preset value for the hour meter time output used on reset
diWaitTime	DINT	Time value for activating the detected error signal
wTypeTime	WORD	Indicating the unit of the displayed operating time at the output Hour

Control Word Bit Description

Functionality

This table describes the control word bits:

Bit Position	Description
0...3	Not used
4	Reset the output q_diHr to value at i_strPara.diSp
5	Pause time counting
6...15	Not used
16	Change in this bit will acknowledge detected error.
17...31	Not used

Status Word

Functionality

This table describes the status word bits:

Bit Position	Description
0	Auto mode is active
1	Manual mode is active
2	No mode is selected
3	Function block is locked by interlock input `i_xLock`
4	Not used
5	Time counting is paused
6	Operational time at `i_strPara.diWaitTime` is exceeded.
7...15	Not used
16	Detected error is reset
17	Detected error is present
18...23	Not used
24	Internal detected error is present
25	External detected error is present
26...31	Not used

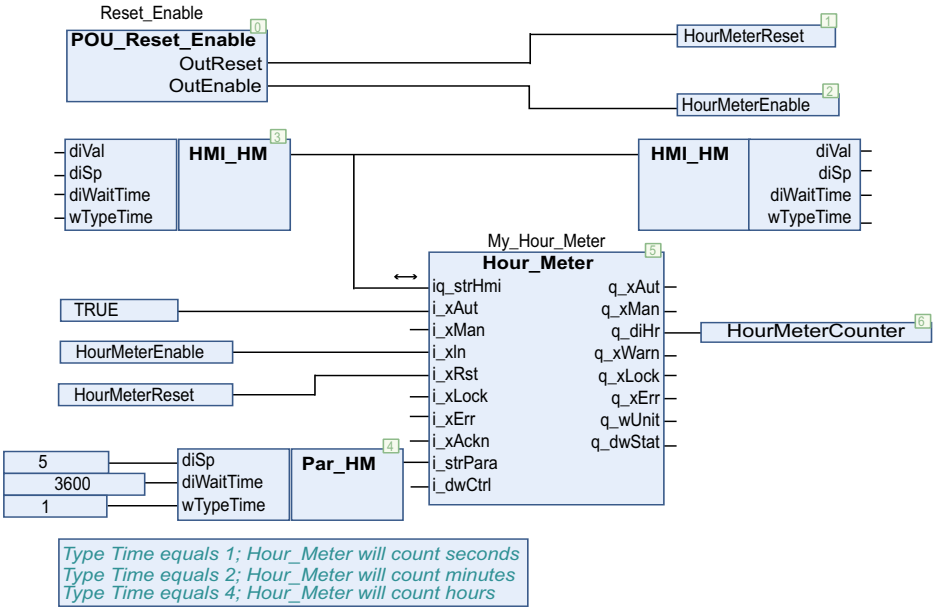
Instantiation and Usage Example

Instantiation and Usage Example

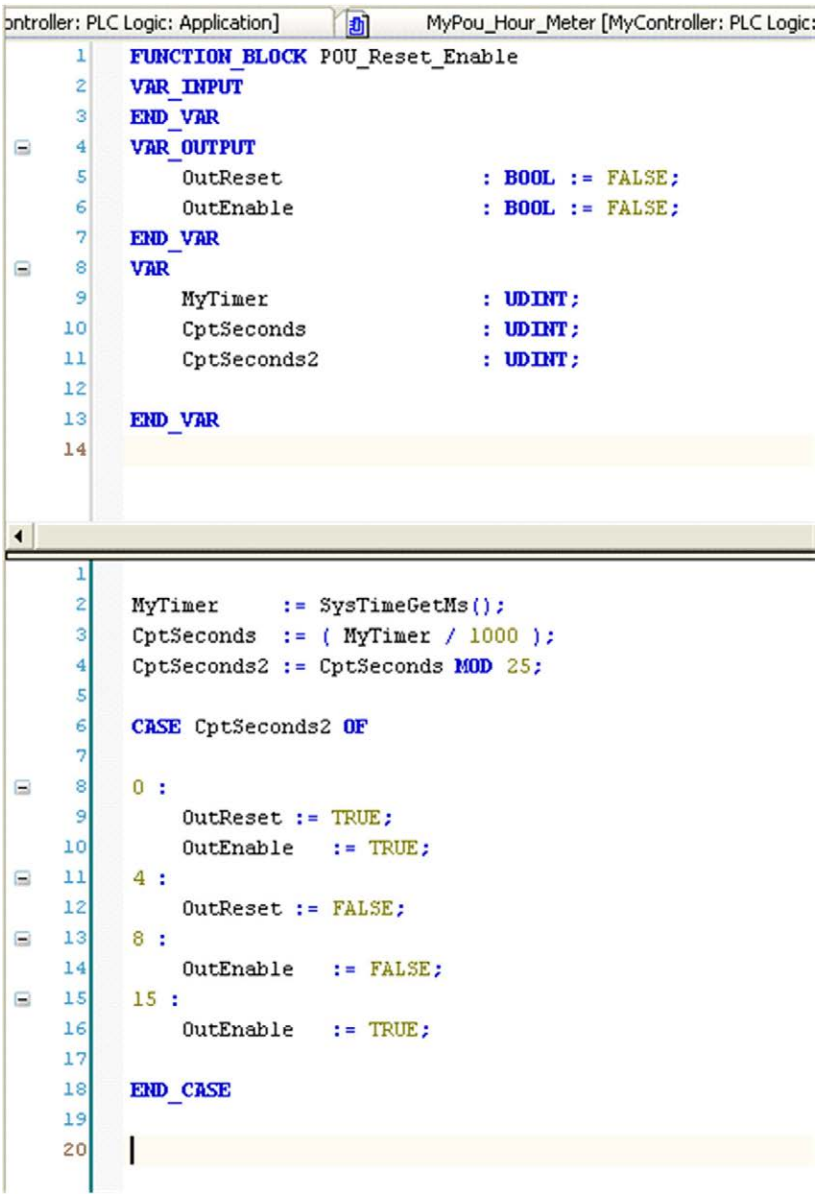
The following program periodically resets and enables the input of `Hour_Meter` function block:

Trace_Hour_Meter [MyController: PLC...] | MyPou_Hour_Meter [MyController: PLC Logic: Application]

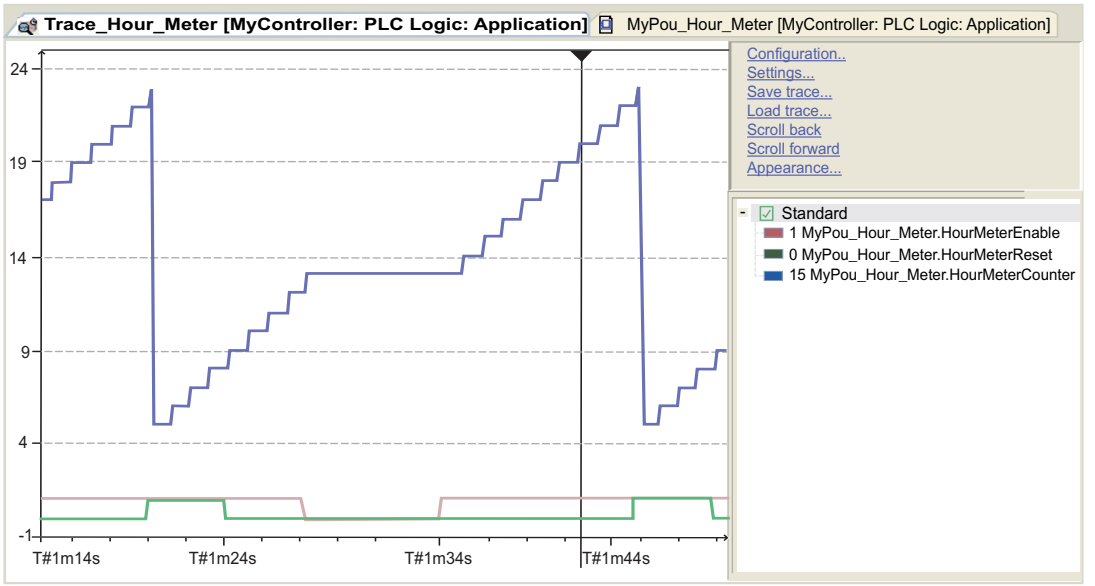
```
1  PROGRAM MyPou_Hour_Meter
2  VAR
3      FlagStart           : BOOL;
4      Reset_Enable        : POU_Reset_Enable;
5      My_Hour_Meter       : Hour_Meter;
6      HourMeterEnable     : BOOL;
7      HourMeterReset      : BOOL;
8      HourMeterCounter    : DINT;
9  END_VAR
```



`HourMeterReset` and `HourMeterEnable` are managed with the following POU:



Every 25 seconds, the data `HourMeterCounter` is reset with an initial value of 5. When `HourMeterReset` is FALSE, the counter holds its current value.



Blue `HourMeterCounter`
Green `HourMeterReset`
Red `HourMeterEnable`

In this sample, the cycle time of the POU in the MAST has no impact. For this sample, the periodicity is 100 milliseconds.

Chapter 50
Operation_Mode: Selecting Operating Mode

Overview

This chapter describes the `Operation_Mode` function block.

What Is in This Chapter?

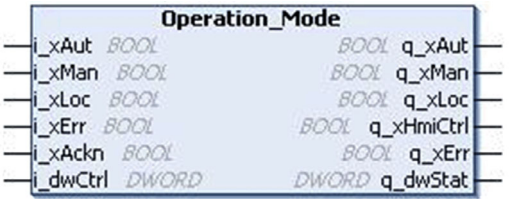
This chapter contains the following topics:

Topic	Page
`Operation_Mode` Function Block	258
Input Pin Description	259
Output Pin Description	260
Control Word Bit Description	261
Status Word	262

Operation_Mode Function Block

Pin Diagram

This figure shows the pin diagram of the Operation_Mode function block:



Functional Description

The Operation_Mode function block is used for selecting between auto and manual operating modes from two different sources:

- Switches / controller program
- HMI

Limitations

- If no operation mode is currently selected, then the previous active mode persists. For example, if auto mode with local control was set previously, then upon resetting the auto input persists till another mode is set.
- The local mode has higher priority than the HMI control. A switch in operating mode does not take place automatically when local mode is reset and HMI control is previously set along with local mode.

Local Mode

The local mode can be activated with auto or manual mode. The local mode is activated by using the input i_xLoc and prohibits manual interaction from the HMI via control word input i_dwCtrl.

With local mode active, the operating mode can be activated by using the inputs i_xAut and i_xMan:

- If i_xAut is set the automatic mode is activated and indicated at the output q_xAut.
- If i_xMan is set the manual mode is activated and indicated at the output q_xMan.

If local mode is not active, then by setting the bits of i_dwCtrl the operating modes can also be activated from the HMI.

NOTE: When q_xHmiCtrl is set, the inputs i_xAut and i_xMan are ignored

Priority

The i_xLoc has a higher priority than the i_dwCtrl command word. So that once the i_xLoc is set, the operating mode is again activated by the inputs i_xAut and i_xMan.

Resetting a Detected Error

The block generates an invalid operation mode, if both auto and manual modes are selected (internal detected error) and displays at output q_xErr. It also sets the detected error signal, if the detected error input i_xErr is set to 1 (external detected error). Detected errors are indicated in the bits of status word q_dwStat. To reset the detected error output the detected error has to be acknowledged by a rising edge on the **input Ack** or by using the acknowledgement bit of the input i_dwCtrl.

Input Pin Description

Input Pin Description

This table describes the input pins of the Operation_Mode function block:

Input	Data Type	Description
i_xAut	BOOL	TRUE: Automatic mode enabled FALSE: Disabled (factory setting)
i_xMan	BOOL	TRUE: Manual mode enabled. FALSE: Disabled. (factory setting)
i_xLoc	BOOL	TRUE: Local mode enabled FALSE: Disabled. (factory setting)
i_xErr	BOOL	TRUE: External detected error is active. FALSE: No external detected error. (factory setting)
i_xAckn	BOOL	Acknowledgement is done with a rising edge. Input to acknowledge internal and external detected errors indicated at the output q_xErr.
i_dwCtrl	DWORD	Command bits to interact from the HMI Range: 0...4294967295 Refer to the status word description *(see page 261)*

Output Pin Description

Output Pin Description

This table describes the output pins of the `Operation_Mode` function block:

Output	Data Type	Description
`q_xAut`	`BOOL`	TRUE: Automatic mode enabled FALSE: Disabled
`q_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled
`q_xLoc`	`BOOL`	TRUE: Local mode enabled The operating mode is not changeable by the HMI and function blocks cannot be operated from the HMI. FALSE: Disabled
`q_xHmiCtrl`	`BOOL`	TRUE: Operating mode is given by the HMI. `q_xHmiCtrl` is overwritten by local mode. FALSE: Disabled
`q_xErr`	`BOOL`	TRUE: Detected error is active. FALSE: No detected error
`q_dwStat`	`DWORD`	Status bits to be displayed in the HMI Range: 0...4294967295 Refer to the status word description *(see page 262)*.

Control Word Bit Description

Functionality

This table describes the control word bits:

Bit Position	Description
0	Set automatic mode
1	Set manual mode
2, 3	Not used
4	Set HMI Control active. Only after this bit is set, selections via bit 0 and bit 1 will take place
5...15	Not used
16	Rising edge of this bit gives acknowledgment to detected error.
17...31	Not used

This table shows the truth table:

i_xAut	i_xMan	i_xLoc	i_dwCtrl			q_xAut	q_xMan	q_xLoc	q_xHmiCtrl	q_xErr
			bit0 Aut	bit1 Man	bit4 HMI					
0	0	0	0	0	0	0	0	0	0	0
0	0	1	X	X	X	0	0	1	0	0
1	1	1	X	X	X	0	0	1	0	1
1	0	1	X	X	X	1	0	1	0	0
0	1	1	X	X	X	0	1	1	0	0
X	X	1 → 0	X	X	0	PS	PS	0	0	0
X	X	0	0	0	1	0	0	0	1	0
X	X	0	1	0	1	1	0	0	1	0
X	X	0	0	1	1	0	1	0	1	0
X	X	0	X	X	1 → 0	PS	PS	0	0	0
X	X	0	1	1	1	0	0	0	1	1
PS Previous State										

Status Word

Functionality

This table describes the status word bits:

Bit Position	Description
0	Auto mode is active
1	Manual mode is active
2	Local mode is active
3	Not used
4	HMI Control mode is active
5...15	Not used
16	Indicates reset of detected error
17	Detected error is present
18...23	Not used
24	Internal detected error is present
25	External detected error is present
26...31	Not used

Part XI
Valve Control

Overview

This part describes the valve control function library.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
51	Bistable_Valve: Controlling the Bistable Valves	265
52	Monostable_Valve: Controlling Monostable Valves	273
53	Proportional_Valve: Controlling Proportional Valves	279

Chapter 51
Bistable_Valve: Controlling the Bistable Valves

Overview

This chapter describes the `Bistable_Valve` function block.

What Is in This Chapter?

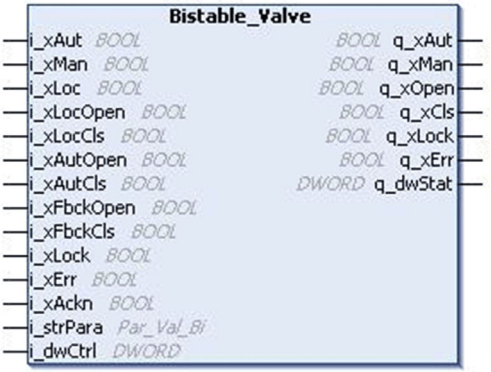
This chapter contains the following topics:

Topic	Page
`Bistable_Valve` Function Block	266
Input Pin Description	268
Output Pin Description	269
Structure Used	269
Control Word Bit Description	270
Status Word	271
Instantiation and Usage Example	272

Bistable_Valve Function Block

Pin Diagram

This figure shows the pin diagram of the Bistable_Valve function block:



Functional Description

The Bistable_Valve function block is used for controlling the bistable valve.

Operation Modes

The Bistable_Valve function block supports three operating modes:

- **Automatic Mode:** The automatic mode is activated by the input pin i_xAut. In this mode, the valve is opened and closed through the inputs i_xAutOpen and i_xAutCls respectively, regardless of the local mode being activated or not.
- **Manual Mode:** The manual mode is activated by the pin i_xMan.
  Case 1: Local mode is not activated. The valve is opened and closed through the bit commands of the signal i_dwCtrl.
  Case 2: Local mode is activated. The valve is opened and closed through the input signals i_xLocOpen and i_xLocCls respectively.
- **Local Mode:** The local mode is activated by an input pin i_xLoc and is set additionally to the automatic or manual mode. The local mode does not influence the automatic mode, but changes the source for manual operation.

Output Behavior

The output q_xOpen remains active as long as the feedback signal i_xFbckOpen remains low. Also, the output q_xCls remains active as long as the feedback signal i_xFbckCls remains low. This output behavior is valid for manual and local modes.

Controller Start Up Behavior

The block is de-activated on controller start and remains in the same operation mode, unless a new one is selected. If both automatic and manual modes are selected simultaneously (inputs `i_xAut` and `i_xMan` are set to 1), the operation mode is invalid which is indicated at the `q_xErr` output.

Supervising the Valve

The position of the valve is supervised by the feedback signals `i_xFbckOpen` and `i_xFbckCls`. Once the operation is started, the feedback inputs must signal the right position of the valve within a defined time. If this time exceeds, then the block indicates a detected error. The time can be set through the structure element iFbckDly at input `i_strPara`.

When both open and close feedback signals are missing (`i_xFbckOpen` and `i_xFbckCls` are set to 0), and the position of the valve is unknown (`QOpen_bi` and `QClose_bi` set to 0), an unknown position error is detected.

When the input `xFbckEn` is FALSE, then the time supervising is NOT enabled. Refer to the Output Behavior *(see page 266)*.

Operating the Valve

The valve can be operated only if the input `i_xLock` is set to 0. An active interlock signal inhibits the operation of the valve. An active interlock is indicated by the output `q_xLock`.

The valve can only be operated if the output `q_xErr` is set to 0. An active detected error signal inhibits the operation of the valve.

Detected Error Management

The output `q_xErr` is high, only if an error is detected. The detected error can be:

- Internal detected error (invalid operation mode, missing feedback signal or unknown position).
- External detected error

The detected errors are indicated in the HMI as alarms. If an interlock or an error is detected during the operation of the valve, the behaviour of the function block depends on the structure element `i_strPara.xFrceEn` at the input `i_strPara`. If this element is set to 1, the block enforces the valve to move in to the default position, and the corresponding output is high (`q_xOpen` or `q_xCls`) for the duration of `i_strPara.iFbckDly` seconds. Otherwise the operation is stopped and has to be restarted after the interlock is gone.

To reset the `q_xErr`, the detected error has to be acknowledged by a rising edge on the input `i_xAckn` or by using bit 16 of the signal `i_dwCtrl`.

Setting Default Position

The default position of the valve can be set by `i_strPara.xPosDflt`. This description assumes Close as the default. If `i_strPara.xPosDflt` is set to 1, Open is the default position.

Input Pin Description

Input Pin Description

This table describes the input pins of the `Bistable_Valve` function block:

Input	Data Type	Description	Remarks
`i_xAut`	`BOOL`	TRUE: Automatic mode enabled FALSE: Disabled	
`i_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled	
`i_xLoc`	`BOOL`	TRUE: Local mode enabled FALSE: Disabled	
`i_xLocOpen`	`BOOL`	Rising edge from 0 to 1 manually opens the valve in local mode	Manual and local mode to be set to 1 simultaneously.
`i_xLocCls`	`BOOL`	Rising edge from 0 to 1 manually closes the valve in local mode.	Manual and local mode to be set to 1 simultaneously.
`i_xAutOpen`	`BOOL`	Rising edge from 0 to 1 opens the valve in automatic mode.	
`i_xAutCls`	`BOOL`	Rising edge from 0 to 1 closes the valve in automatic mode.	
`i_xFbckOpen`	`BOOL`	TRUE: Open feedback signal is active. FALSE: No open feedback	
`i_xFbckCls`	`BOOL`	TRUE: Close feedback signal is active. FALSE: No close feedback	
`i_xLock`	`BOOL`	TRUE: Interlock is active FALSE: No interlock (Optional)	
`i_xErr`	`BOOL`	TRUE: External detected error is active. FALSE: No external detected error	
`i_xAckn`	`BOOL`	Acknowledgement is done with a rising edge.	Input to acknowledge internal and external detected errors indicated at the output `q_xErr`.
`i_strPara`	`STRUCT` `Par_Val_Bi`	Structure with parameters for this block.	Refer to the used structures *(see page 269)* description.
`i_dwCtrl`	`DWORD`	Command bits to interact from the HMI Range: 0...4294967295	Refer to the control word *(see page 270)* description

Output Pin Description

Output Pin Description

Output Pin Description

This table describes the output pins of the `Bistable_Valve` function block:

Output	Data Type	Description	Remarks
`q_xAut`	`BOOL`	TRUE: Automatic mode enabled FALSE: Disabled	-
`q_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled	-
`q_xOpen`	`BOOL`	TRUE: Open command active FALSE: Disabled	-
`q_xCls`	`BOOL`	TRUE: Close command active FALSE: Disabled	-
`q_xLock`	`BOOL`	TRUE: Interlock is active FALSE: No interlock	Indicates that the operation is blocked by an interlock (input `i_xLock`)
`q_xErr`	`BOOL`	TRUE: Detected error is active FALSE: No detected error)	-
`q_dwStat`	`DWORD`	Status bits to be displayed in the HMI Range: 0...4294967295	Refer to the status word description *(see page 271)*.

Structure Used

`Par_Val_Bi`

Structure Element	Type	Description
`xFbckEn`	`BOOL`	Enable feedback signal supervision
`iFbckDly`	`INT`	Delay time in seconds to get the feedback signal from the valve.
`iRevDly`	`INT`	Delay time in seconds to operate the valve in opposite direction.
`xPosDflt`	`BOOL`	Default position of the valve (FALSE: closed, TRUE: opened).
`xFrceEn`	`BOOL`	Enable enforcement of default position at interlock or detected error.

Control Word Bit Description

Functionality

This table describes the control word bits:

Bit Position	Description
0...7	Not used
8	Rising edge from 0 to 1 manually opens the valve (output `q_xOpen` set to TRUE) in manual mode.
9	Rising edge from 0 to 1 manually closes the valve (output `q_xCls` set to TRUE) in manual mode.
10...15	Not used
16	Acknowledges internal and external detected errors indicated at the output `q_xErr`. This is rising edge triggered.
17...31	Not used

Status Word

Functionality

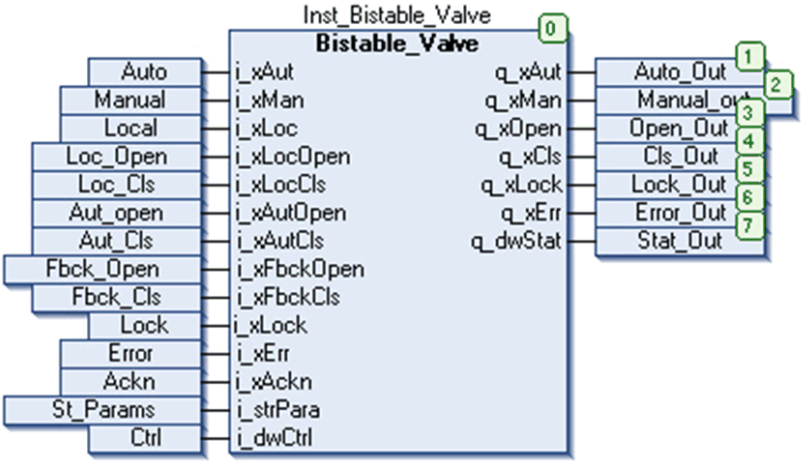
This table describes the status word:

Bit Position	Description
0	Auto mode is active.
1	Manual mode is active.
2	Local mode is selected.
3	Indicates that the operation is blocked by an interlock (input `i_xLock`).
4...7	Not used
8	Signal to open the valve.
9	Signal to close the valve.
10...13	Not used
14	Feedback signal from opened valve.
15	Feedback signal from closed valve.
16	Resets the detected error (`q_xErr`).
17	Indicates that the operation is blocked by an internal or external (Input `i_xErr`) detected error, which is not acknowledged.
18	Not used
19	Enable feedback signal supervision.
20	Default Position of the valve (FALSE: closed, TRUE: opened).
21...23	Not used
24	Invalid Operating mode detected error.
25	External detected error.
26	Missing feedback detected error.
27	Unknown position detected error.
28...31	Not used

Instantiation and Usage Example

Instantiation and Usage Example

This figure shows an instance of the `Bistable_Valve` function block:



Limitations

When forced enable is active (`xFrceEn`), the valve is forced in to the default position (`xPosDflt`) only for the time `iFbckDly` seconds. It is reset if an appropriate feedback signal is activated or interlock signal is removed or the `q_xErr` output is acknowledged.

Ensure that appropriate feedback signal is activated before executing the block; else an unknown position error is detected after a time delay of `iFbckDly` seconds.

Chapter 52
Monostable_Valve: Controlling Monostable Valves

Overview

This chapter describes the `Monostable_Valve` function block.

What Is in This Chapter?

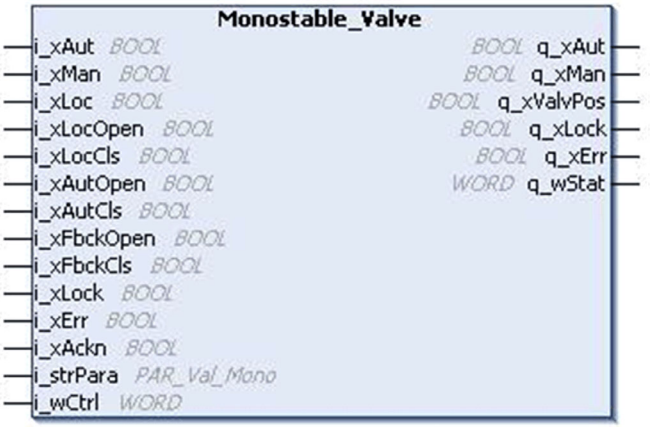
This chapter contains the following topics:

Topic	Page
`Monostable_Valve` Function Block	274
Input Pin Description	276
Output Pin Description	277
Structure Used	277
Control Word Bit Description	278
Status Word	278

Monostable_Valve Function Block

Pin Diagram

This figure shows the pin diagram of the Monostable_Valve function block:



Functional Description

The Monostable_Valve function block is used for controlling the monostable valve.

Operation Modes

The Monostable_Valve function block supports three operating modes:

- **Automatic Mode:** The automatic mode is activated by the input pin i_xAut. In this mode the valve is opened and closed through the inputs i_xAutOpen (default position is Closed) and i_xAutCls (default position is Open) respectively, regardless of the local mode being activated or not. The output q_xValvPos remains active as long as the inputs i_xAutOpen/i_xAutCls remain active.
- **Manual Mode:** The manual mode is activated by the pin i_xMan.
  Case 1: Local mode is not activated. The valve is opened and closed through the bit commands of the signal i_dwCtrl.
  Case 2: Local mode is activated. The valve is opened and closed through the input signals i_xLocOpen and i_xLocCls respectively.
- **Local Mode:** Local mode is activated by an input pin i_xLoc and is set additionally to automatic or manual mode. Local mode does not influence the automatic mode, but changes the source for manual operation.

NOTE: If both automatic and manual modes are selected simultaneously (inputs i_xAut and i_xMan are set to 1), the operation mode is invalid, which is indicated at the q_xErr output.

Controller Start Up Behavior

The block is de-activated on controller start and remains in the same operation mode, unless a new one is selected.

Supervising the Valve

The position of the valve is supervised by the feedback signals `i_xFbckOpen` and `i_xFbckCls`. Once the operation is started, the feedback inputs must signal the right position of the valve within a defined time. If this time exceeds, then the block indicates a detected error (missing feedback detected error).The time can be set through the structure element `iFbckDly` at input `i_strPara`. The supervision can be switched Off by the structure element `xFbckEn` at the input `i_strPara`.

Operating the Valve

The valve can be operated only if the input `i_xLock` is set to 0. An active interlock signal inhibits the operation of the valve and is indicated by the output pin `q_xLock`.

The valve can only be operated if the output `q_xErr` is set to 0. An active detected error signal inhibits the operation of the valve.

Detected Error Management

The output `q_xErr` is high, only if an error is detected. The detected error can be:

- Internal detected error (invalid operation mode, missing feedback signal or unknown position).
- External detected error

Detected errors are indicated in the HMI as alarms. If an interlock or an error is detected during the operation of the valve, the behaviour of the function block depends on the structure element `i_strPara.xFrceEn` at input `i_strPara`. If this element is set to 1, the block enforces the valve to move in to the default position, and the corresponding output is high (`q_xOpen` or `q_xCls`) for the duration of `i_strPara.iFbckDly` seconds. Otherwise the operation is stopped and has to be restarted after the interlock is gone.

To reset `q_xErr`, the detected error has to be acknowledged by a rising edge on the input `i_xAckn` or by using bit 16 of the signal `i_dwCtrl`.

Setting Default Position

The default position of the valve can be set by `i_strPara.xPosDflt`. This description assumes Close as default. If `i_strPara.xPosDflt` is set to 1, Open is the default position.

Input Pin Description

Input Pin Description

This table describes the input pins of the `Monostable_Valve` function block:

Input	Data Type	Description	Remarks
`i_xAut`	`BOOL`	TRUE: Automatic mode enabled FALSE: Disabled	-
`i_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled	-
`i_xLoc`	`BOOL`	TRUE: Local mode enabled FALSE: Disabled	-
`i_xLocOpen`	`BOOL`	Rising edge from 0 to 1 manually opens the valve in local mode.	Manual and Local Mode to be set to 1 simultaneously.
`i_xLocCls`	`BOOL`	Rising edge from 0 to 1 manually closes the valve in local mode.	Manual and Local Mode to be set to 1 simultaneously.
`i_xAutOpen`	`BOOL`	TRUE: Valve open command is enabled. FALSE: Disabled	-
`i_xAutCls`	`BOOL`	TRUE: Valve close command is enabled. FALSE: Disabled	-
`i_xFbckOpen`	`BOOL`	TRUE: Open feedback signal is active. FALSE: No open feedback	-
`i_xFbckCls`	`BOOL`	TRUE: Close feedback signal is active. FALSE: No close feedback	-
`i_xLock`	`BOOL`	TRUE: Interlock is active FALSE: No interlock	Interlock input for valve operation. Valve operation is inhibited, when the input is set to 1.
`i_xErr`	`BOOL`	TRUE: External detected error is active. FALSE: No external detected error	-
`i_xAckn`	`BOOL`	Acknowledgement is done with a rising edge.	Input to acknowledge internal and external detected errors indicated at the output `q_xErr`.
`i_strPara`	`STRUCT PAR_Val_Mono`	Structure with parameters for this block.	Refer to the used structure *(see page 277)* description.
`i_wCtrl`	`WORD`	Command bits to interact from the HMI. Range: 0...65535	Refer to the command word description *(see page 278)*.

Output Pin Description

Output Pin Description

Output Pin Description

This table describes the output pins of the `Monostable_Valve` function block:

Identifiers	Output Type	Description	Remarks
`q_xAut`	`BOOL`	TRUE: Automatic mode enabled FALSE: Disabled	-
`q_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled	-
`q_xValvPos`	`BOOL`	TRUE: Open/Close command active FALSE: Disabled	Open command active if default position is closed/Close command active if default position is open.
`q_xLock`	`BOOL`	TRUE: Interlock is active. FALSE: No interlock	Indicates that the operation is blocked by an interlock (input `i_xLock`)
`q_xErr`	`BOOL`	TRUE: Detected error is active. FALSE: No detected error	-
`q_wStat`	`WORD`	Status bits to be displayed in the HMI Range: 0...65535	Refer to the status word description *(see page 278)*.

Structure Used

`PAR_Val_Mono`

Structure Element	Type	Description
`xFbckEn`	`BOOL`	Enable feedback signal supervision
`iFbckDly`	`INT`	Delay time in seconds to get the feedback signal from Valve
`xPosDflt`	`BOOL`	Default position of the valve (0: closed, 1: opened)

Control Word Bit Description

Functionality

This table describes the control word bits:

Bit Position	Description
0	Rising edge from 0 to 1 manually opens the valve in manual mode.
1	Rising edge from 0 to 1 manually closes the valve in manual mode.
2	Input to acknowledge internal and external detected errors indicated at the output `q_xErr`. Acknowledgement is done with a rising edge
3...15	Not used

Status Word

Functionality

This table describes the status word:

Bit position	Description
0	Auto mode is active.
1	Manual mode is active.
2	Local mode is active.
3	Valve is opened/closed depending upon the default position.
4	Default position of the Valve.
5	Interlock signal is active.
6	Feedback signal from opened valve.
7	Feedback signal from closed valve.
8	Detected error is active.
9	Invalid operating mode detected error.
10	External detected error.
11	Missing feedback detected error.
12	Enable feedback supervision.
13-15	Not used.

Chapter 53
Proportional_Valve: Controlling Proportional Valves

Overview

This chapter describes the `Proportional_Valve` function block.

What Is in This Chapter?

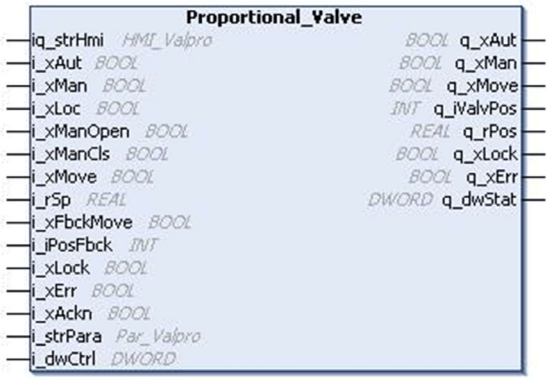
This chapter contains the following topics:

Topic	Page
`Proportional_Valve` Function Block	280
Input Pin Description	282
Output Pin Description	283
Input - Output Pin	284
Structures Used	284
Control Word Bit Description	285
Status Word	286
Instantiation and Usage Example	287

Proportional_Valve Function Block

Pin Diagram

This figure shows the pin diagram of the Proportional_Valve function block:



Functional Description

The Proportional_Valve function block is used for controlling proportional valve.

Operation Modes

The Proportional_Valve function block supports three operating modes:

- **Automatic Mode:** The automatic mode is activated by the input pin i_xAut. In this mode, the valve is opened and closed via the input i_xMove, regardless if the local mode is activated or not. The new set point is given at the input i_rSp.
- **Manual Mode:** The manual mode is activated by the pin i_xMan.
  Case 1: Local mode is not activated. The valve is opened and closed through a bit command in the variable i_dwCtrl and the value of the set point is given by rSP on the input-output iq_strHmi
  Case 2: Local mode is activated. The valve is operated through the inputs i_xManOpen and i_xManCls.
- **Local Mode:** The local mode is activated by an input pin i_xLoc and is set additionally to the automatic or manual mode. The local mode does not influence the automatic mode, but changes the source for manual operation. The output q_iValvPos is automatically set to i_strPara.rMinSp when using i_xManCls or i_strPara.rMaxSp when using i_xManOpen.

NOTE: If the operation mode is changed from manual or HMI mode to automatic mode, a movement of the valve is stopped. Any other change of the operation mode does not affect the valve movement, but the setpoint is adjusted to the value of the actual operation mode.

Output Behavior

The output q_xMove remains active as long as the new position given by the setpoint is not reached.

Setting a Deadband

A deadband can be set by i_strPara.rBnd, so that q_xMove is switched Off, when the deviation of actual position and setpoint is less than the deadband.

When using the inputs i_xManOpen and i_xManCls in local mode, the output q_xMove is active as long as the inputs are active or if the maximum or minimum position is reached (taking dead band into consideration).

Supervising the Valve

The position of the valve is supervised by the feedback signals i_xFbckOpen and i_xFbckCls. Once the operation is started, the feedback inputs must signal the right position of the valve within a defined time. If this time exceeds, then the block indicates a detected error. The time can be set through the structure element iFbckDly at input i_strPara. The supervision can be switched Off by the structure element xFbckEn at the input i_strPara.

Operating the Valve

The valve can be operated if the input i_xLock is set to 0. An active interlock signal inhibits the operation of the valve. An active interlock is indicated by the output q_xLock.

The valve can only be operated, if the output q_xErr is set to 0. An active detected error signal inhibits the operation of the valve.

Detected Error Management

The output q_xErr is high if an error is detected. The detected error can be:

- Internal detected error (invalid operation mode, missing feedback signal or unknown position.
- External detected error

The detected errors are indicated in the HMI as alarms. If an interlock or an error is detected during the operation of the valve, the behaviour of the function block depends on the structure element i_strPara.xFrceEn at input i_strPara. If this element is set to 1, the block enforces the valve to move in to the default position, and the output is high (q_xMove) for the duration of i_strPara.iFbckDly seconds. Otherwise the operation is stopped and has to be restarted after the interlock is gone.

To reset the q_xErr, the detected error has to be acknowledged by a rising edge on the input i_xAckn or by using bit 16 of the signal i_dwCtrl.

Setting Default Position

The default position of the valve can be set by `i_strPara.xPosDfltSet`. This description assumes Close as default. If `i_strPara.xPosDflt` is set to 1, Open is the default position.

Input Pin Description

Input Pin Description

This table describes the input pins of the `Proportional_Valve` function block:

Input	Data Type	Description	Remarks
`i_xAut`	`BOOL`	TRUE: Automatic mode enabled FALSE: Disabled	-
`i_xMan`	`BOOL`	TRUE: Manual mode enabled FALSE: Disabled	-
`i_xLoc`	`BOOL`	TRUE: Local mode enabled FALSE: Disabled	-
`i_xManOpen`	`BOOL`	TRUE: Manually opens the valve (output `q_xMove`) in local mode as long as the signal is activated. FALSE: Disabled	-
`i_xManCls`	`BOOL`	TRUE: Manually closes the valve (output `q_xMove`) in local mode as long as the signal is activated. FALSE: Disabled	-
`i_xMove`	`BOOL`	TRUE: Move the valve into a new position in automatic mode FALSE: Disabled	-
`i_rSp`	`REAL`	New position for movement of valve in automatic mode Range: $1.17e^{-38}$...$3.4e^{+38}$	-
`i_xFbckMove`	`BOOL`	TRUE: Feedback for movement of the valve FALSE: Disabled	-
`i_iPosFbck`	`INT`	Feedback of Valve position. Range: 0...31	-
`i_xLock`	`BOOL`	TRUE: Interlock enabled. FALSE: Disabled	Interlock input for valve operation. Valve operation is inhibited, when the input is set to 1.
`i_xErr`	`BOOL`	TRUE: External detected error is active. FALSE: No external detected error	-
`i_xAckn`	`BOOL`	Acknowledgement is done with a rising edge.	Input to acknowledge internal and external detected errors indicated at the output `q_xErr`.

Input	Data Type	Description	Remarks
i_strPara	STRUCT Par_Valpro	Structure with parameters for this block.	Refer to the used structure descriptions *(see page 284)*.
i_dwCtrl	DWORD	Command bits to interact from the HMI. Range: 0...4294967295	Refer to the command word descriptions *(see page 285)*.

Output Pin Description

Output Pin Description

This table describes the output pins of the Proportional_Valve function block:

Identifiers	Output Type	Description	Remarks
q_xAut	BOOL	TRUE: Automatic mode enabled FALSE: Disabled	-
q_xMan	BOOL	TRUE: Manual mode enabled FALSE: Disabled	-
q_xMove	BOOL	TRUE: Open Command Active FALSE: Disabled	-
q_iValvPos	INT	Signal to indicate the new position of the valve. The value is given by the set point in automatic or manual mode. If the valve is operated manually in local mode the value is set to completely opened or completely closed. Range: 0...31	If enforcement of default position is active, the signal will be set to the default position of the valve in case of interlock or detected error.
q_rPos	REAL	Actual position of the valve. Range: $\pm 3.4e^{+38}$	-
q_xLock	BOOL	TRUE: Interlock is active. FALSE: No interlock	Indicates that the operation is blocked by an interlock (input i_xLock)
q_xErr	BOOL	TRUE: Detected error is active. FALSE: No detected error	-
q_dwStat	DWORD	Status bits to be displayed in the HMI Range: 0...4294967295	Refer to the status word description *(see page 286)*.

Input - Output Pin

Description

This table describes the `iq_strHmi` input/output:

Identifier	Type	Description
`iq_strHmi`	`STRUCT HMI_Valpro`	Structure to interface with the HMI Refer to used structures *(see page 284)*.

Structures Used

`Par_Valpro`

Structure Element	Type	Description
`xFbEn`	`BOOL`	Enable feedback signal supervision
`iFbTime`	`INT`	Delay time in seconds to get the feedback signal and valve to reach the set point
`xPosDflt`	`BOOL`	Default Position of the valve (0: closed, 1: opened)
`xPosDfltSet`	`BOOL`	Enable enforcement of default position at interlock or detected error
`rMinSp`	`REAL`	Minimum value for setpoint: close position
`rMaxSp`	`REAL`	Maximum value for setpoint: open position
`rCnvrFact`	`REAL`	Conversion factor for position output and feedback position input
`rBnd`	`REAL`	Allowed deviation between setpoint and position

`HMI_Valpro`

Structure Element	Type	Description
`rVal`	`REAL`	Actual position of the valve
`rSp`	`REAL`	Setpoint for manual mode
`rHighLim`	`REAL`	Low limit for generating a detected error and alarm
`rLowLim`	`REAL`	High limit for generating a detected error and alarm

Control Word Bit Description

Functionality

This table describes the control word bits:

Bit Position	Description
0...9	Not used
10	Move the valve towards the set point. Bit is Edge Triggered
11...15	Not used
16	Rising edge of this bit gives acknowledgment to detected error.
17...31	Not used

NOTE: On receiving an Edge trigger (1->0 or 0->1) at this bit(Control Word bit 10), Move command to the valve is active and is SET to TRUE as long as the valve does not reach set point value. Move command becomes inactive and is RESET to FALSE once the valve reaches the set point value and is monitored using `i_iPosFbck` input OR if `q_xErr` is TRUE.

Status Word

Functionality

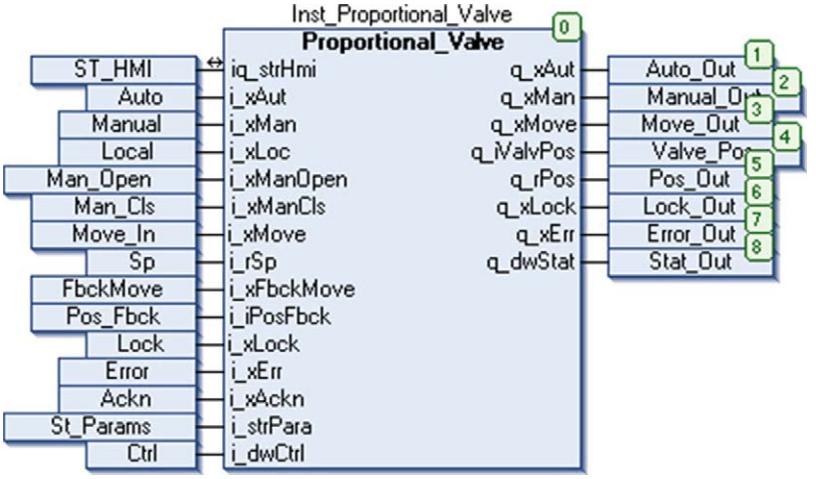
This table describes the status word:

Bit Position	Description
0	Auto mode is active.
1	Manual mode is active.
2	Local mode is active.
3	Function block is locked by interlock input `i_xLock`.
4...7	Not used
8	Maximum set point reached by valve(read at input `i_iPosFbck`.
9	Minimum set point reached by valve(read at input `i_iPosFbck`.
10	Indicates movement of valve.
11...13	Not used
14	This bit is set to 1, if `i_strPara.xFbEn` is true and there is a Valve move feedback or if `i_strPara.xFbEn` is false and there is a Valve move command
15	Not used
16	Indicates reset of detected error.
17	Detected error is present.
18	Not used
19	Equivalent to `i_strPara.xFbEn` (Feedback enable).
20	Equivalent to `i_strPara.xPosDflt` (Default position).
21...23	Not used
24	Invalid operating mode.
25	External detected error is present.
26	Feedback missing.
27	Valve has reached the low limit specified at `i_strPara.rMinSp`.
28	Valve has reached the high limit specified at `i_strPara.rMaxSp`.
29...31	Not used

Instantiation and Usage Example

Instantiation and Usage Example

This figure shows an instance of the `Proportional_Valve` function block:



Limitations

If feedback is enabled by setting `i_strPara.xFbEn` = 1, then not only the movement feedback has to come at the input `i_xFbckMove` but also the valve has to reach the setpoint within the specified time `i_strPara.iFbTime`.

If `i_strPara.rMinSp` = `i_strPara.rMaxSp` = `i_rSP` and the block is set in auto mode, then both the bits of status word `q_dwStat` for valve open and valve close is set to 1.

While operating with feedback enabled, even if no move feedback is received by the block but the valve reaches setpoint within the specified time, then no error is detected.

Priority Management Limitations

While operating in manual mode with local control, inputs `i_xManOpen` and `i_xManCls` given together does not produce any result until one of them is withdrawn. However in this case, the value of output `q_iValvPos` is unpredictable as long as both inputs are high.

Glossary

!

%

According to the IEC standard, % is a prefix that identifies internal memory addresses in the logic controller to store the value of program variables, constants, I/O, and so on.

A

analog input

Converts received voltage or current levels into numerical values. You can store and process these values within the logic controller.

analog output

Converts numerical values within the logic controller and sends out proportional voltage or current levels.

ARRAY

The systematic arrangement of data objects of a single type in the form of a table defined in logic controller memory. The syntax is as follows: `ARRAY [<dimension>] OF <Type>`

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

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

ARW

(*anti-reset windup*) The feature that shuts off the reset action when the measurement is outside the proportional band to help prevent the reset circuit from overloading. Reset action begins again when the measurement returns to within the proportional band. Anti-reset windup is standard in most quality PID controllers.

ASCII

(*American standard code for Information Interchange*) A protocol for representing alphanumeric characters (letters, numbers, certain graphics, and control characters).

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.

closed loop

A closed loop control is a motion control system that used both positional feedback and velocity feedback to generate a correction signal. It does this by comparing its position and velocity to the values of specified parameters. The devices providing the feedback are typically encoders, resolvers, LVTDs, and tachometers.

See also: *open loop*

control network

A network containing logic controllers, SCADA systems, PCs, HMI, switches, ...

Two kinds of topologies are supported:

- flat: all modules and devices in this network belong to same subnet.
- 2 levels: the network is split into an operation network and an inter-controller network.

These two networks can be physically independent, but are generally linked by a routing device.

cyclic task

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

D

DWORD

(*double word*) Encoded in 32-bit format.

E

element

The short name of the ARRAY element.

equipment

A part of a machine including sub-assemblies such as conveyors, turntables, and so on.

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)

H

HMI

(*human machine interface*) An operator interface (usually graphical) for human control over industrial equipment.

I

I/O

(*input/output*)

ID

(*identifier/identification*)

input/output

The index of the ARRAY.

M

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.

ms

(*millisecond*)

N

nibble

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

O

open loop

Open loop control refers to a motion control system with no external sensors to provide position or velocity correction signals.

See also: *closed loop*.

P

PID

(*proportional, integral, derivative*) A generic control loop feedback mechanism (controller) widely used in industrial control systems.

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.

PWM

(*pulse width modulation*) A fast output that oscillates between off and on in an adjustable duty cycle, producing a rectangular wave form (though you can adjust it to produce a square wave).

S

scan

A function that includes:

- reading inputs and placing the values in memory
- executing the application program 1 instruction at a time and storing the results in memory
- using the results to update outputs

setpoint

In a PID controller, the target value set by the user. The main objective of the PID controller is to ensure that the process value reaches the setpoint.

See also *process value*.

STOP

A command that causes the controller to stop running an application program.

string

A variable that is a series of ASCII characters.

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

A
Analysis, *179*
ArrayOfByte_TO_String, *217*

B
Bistable_Valve, *265*
bit organization for DWORD, *25*

C
Celsius_TO_Fahrenheit, *233*
Celsius_TO_Kelvin, *235*
Check_Divisor, *213*

D
DT_AS_WORD, *221*
DWORD_AS_WORD, *223*

F
Fahrenheit_TO_Celsius, *237*
FB_2points, *33*
FB_3points, *39*
FB_3points_Ext, *45*
FB_Cyclic_Monitoring, *95*
FB_DeadBand, *101*
FB_Limiter, *105*
FB_P, *53*
FB_PI, *59*
FB_PI_PID, *85*
FB_PID, *67*
FB_PWM, *109*
FB_Redundant_Sensor_Monitoring, *119*
FB_Scaling, *127*
FB_Sensor_Monitoring, *131*
Filter_AnalogInput, *141*
Filter_Arithmetic, *145*
Filter_MovingAverage, *149*
Filter_PT1, *153*
Frequency_Multiplier, *181*
Frequency_Output, *187*
Frequency_TO_Period, *239*

G
GPL
    Toolbox library, *139*

H
Hour_Meter, *247*

J
JK_FlipFlop, *165*
JK_FlipFlop_MasterSlave, *167*

K
Kelvin_TO_Celsius, *241*

M
Monostable_Valve, *273*

N
Normalizer_With_Limiter, *195*

O
ONE_SEC_PULSE, *199*
Operation_Mode, *257*

P

Period_TO_Frequency, *243*
PI, *31*
PID, *31*
Proportional_Valve, *279*

Q

Quantizer, *201*

R

RS_FlipFlop, *171*

S

SetBitTo, *27*
Signal_Saturation, *203*
Signal_Statistics, *209*
SR_FlipFlop, *173*
String_TO_ArrayOfByte, *225*
system requirements, *21*

T

TestBit, *29*
Toggle_FlipFlop, *175*
Toolbox
  Analysis, *179*
  ArrayOfByte_TO_String, *217*
  Bistable_Valve, *265*
  bit organization for DWORD, *25*
  Celsius_TO_Fahrenheit, *233*
  Celsius_TO_Kelvin, *235*
  Check_Divisor, *213*
  DT_AS_WORD, *221*
  DWORD_AS_WORD, *223*
  Fahrenheit_TO_Celsius, *237*
  FB_2points, *33*
  FB_3points, *39*
  FB_3points_Ext, *45*
  FB_Cyclic_Monitoring, *95*
  FB_DeadBand, *101*
  FB_Limiter, *105*
  FB_P, *53*
  FB_PI, *59*
  FB_PI_PID, *85*
  FB_PID, *67*
  FB_PWM, *109*
  FB_Redundant_Sensor_Monitoring, *119*
  FB_Scaling, *127*
  FB_Sensor_Monitoring, *131*
  Filter_AnalogInput, *141*
  Filter_Arithmetic, *145*
  Filter_MovingAverage, *149*
  Filter_PT1, *153*
  Frequency_Multiplier, *181*
  Frequency_Output, *187*
  Frequency_TO_Period, *239*

Hour_Meter, *247*
JK_FlipFlop, *165*
JK_FlipFlop_MasterSlave, *167*
Kelvin_TO_Celsius, *241*
Monostable_Valve, *273*
Normalizer_With_Limiter, *195*
ONE_SEC_PULSE, *199*
Operation_Mode, *257*
Period_TO_Frequency, *243*
Proportional_Valve, *279*
Quantizer, *201*
RS_FlipFlop, *171*
SetBitTo, *27*
Signal_Saturation, *203*
Signal_Statistics, *209*
SR_FlipFlop, *173*
String_TO_ArrayOfByte, *225*
system requirements, *21*
TestBit, *29*
Toggle_FlipFlop, *175*
WORD_AS_DWORD, *229*
Toolbox library
GPL, *139*

W

WORD_AS_DWORD, *229*

EcoStruxure Machine Expert

Manage a Cyclic Task Interval Toolbox_Advance

Library Guide

06/2019

EIO0000000946.02

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

	Safety Information	5
	About the Book	9
Chapter 1	Presentation	13
	Presentation	13
Chapter 2	Functions Descriptions	13
	GetCurrentTaskCycle: Gets the Current Cyclic Task Interval	14
	SetCurrentTaskCycle: Sets the Cyclic Task Interval	16
Appendices		13
Appendix A	Function and Function Block Representation	13
	Differences Between a Function and a Function Block	14
	How to Use a Function or a Function Block in IL Language	15
	How to Use a Function or a Function Block in ST Language	18
Glossary		31
Index		33

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

BEFORE YOU BEGIN

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

⚠ WARNING

UNGUARDED EQUIPMENT

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

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

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

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

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

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

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

START-UP AND TEST

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

⚠ WARNING

EQUIPMENT OPERATION HAZARD

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

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

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

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

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

Before energizing equipment:

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

OPERATION AND ADJUSTMENTS

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

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

About the Book

At a Glance

Document Scope

This documentation describes the Toolbox_Advance library functions used to get and set the interval of a Cyclic Task.

Validity Note

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

Related Documents

Title of the Document	Reference Number
Modicon M258 Logic Controller Programming Guide	EIO0000000402 (ENG) EIO0000000403 (FRE) EIO0000000404 (GER) EIO0000000405 (SPA) EIO0000000406 (ITA) EIO0000000407 (CHS)
Modicon LMC058 Logic Controller Programming Guide	EIO0000000408 (ENG) EIO0000000409 (FRE) EIO0000000410 (GER) EIO0000000411 (SPA) EIO0000000412 (ITA) EIO0000000413 (CHS)
Magelis XBTGC HMI Controller Programming Guide	EIO0000000650 (ENG) EIO0000000651 (FRE) EIO0000000652 (GER) EIO0000000653 (SPA) EIO0000000654 (ITA) EIO0000000655 (CHS)
Magelis XBTGT, XBTGK HMI Controller Programming Guide	EIO0000000638 (ENG) EIO0000000639 (FRE) EIO0000000640 (GER) EIO0000000641 (SPA) EIO0000000642 (ITA) EIO0000000643 (CHS)

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.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

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

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.

Chapter 1
Presentation

Presentation

Overview

The Toolbox_Advance library offers 2 functions that can be used in a Program Organisation Unit (POU) to get (`GetCurrentTaskCycle`) and set (`SetCurrentTaskCycle`) the interval of the attached Cyclic Task.

NOTE: These functions can be used in POUs attached to Cyclic Tasks only. For further information about the Tasks configuration and POUs attachment, refer to the Tasks chapter of your Controller Programming Guide.

Adding the Toolbox_Advance Library

To have access to the Toolbox_Advance functions, it is necessary to manually add the library:

Step	Action
1	Double-click the **Library Manager** node of the controller **Application** node in the **Devices** window.
2	Click **Add Library**. Result: The **Add Library** dialog box opens.
3	Browse to **Util**, visible when the **Company** filter is set to **Schneider Electric** or **All Company**. Result: The **Util** libraries list is displayed.
4	Select the **Toolbox_Advance library** and press **OK**. Result: The library is added to the **Library Manager** list.

Chapter 2
Functions Descriptions

Overview

This chapter describes the functions of the Toolbox_Advance library.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
GetCurrentTaskCycle: Gets the Current Cyclic Task Interval	14
SetCurrentTaskCycle: Sets the Cyclic Task Interval	16

GetCurrentTaskCycle: Gets the Current Cyclic Task Interval

Function Description

This function returns an operation diagnostic and the current interval of the task attached Cyclic Task in microseconds (µs).

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 13)*.

I/O Variables Description

The following table describes the output variable:

Output	Type	Description
`GetCurrentTaskCycle`	`UDINT`	Function operation diagnostic: 0 = No error detected or Non Cyclic Task 2 = Invalid parameter detected 12 = Function not implemented in the controller

The following table describes the input/output variable:

Input/Output	Type	Description
`currentTime`	`UDINT`	Current interval of the POU attached Task in microseconds (µs). **NOTE:** Returns 0 if an error has been detected.

NOTE: If this function is called in POU attached to a non Cyclic Task, the function returns 0 and `currentTime` is set to 0.

Example

The following example describes how to get the current interval of the Cyclic Task attached to a POU `GetTaskInterval` in ST language. If a function operation error is detected, a diagnostic flag is set to `TRUE` and the error code is stored.

If no error detected, the diagnostic flag is set to `FALSE` and the valid task interval is stored.

Variable declaration:

```
PROGRAM GetTaskInterval
VAR
   // Last Task interval
   TaskInterval: UDINT := 0;
   // Last valid Task interval
   TaskInterval_Memo: UDINT := 0;
   // GetCurrentTaskCycle function operation diagnostic
   GetTaskCycle_Diag: UDINT := 0;
   // Memorisation of last GetCurrentTaskCycle detected error
   code
   GetTaskCycle_Diag_Memo: UDINT := 0;
   // GetCurrentTaskCycle error detected flag
   GetTaskCycle_Err: BOOL := FALSE;
END_VAR
```

Program:

```
// Get the last Cyclic Task interval
GetTaskCycle_Diag:=GetCurrentTaskCycle(TaskInterval);
// Check diagnostic
IF TaskInterval=0 OR GetTaskCycle_Diag<>0
   THEN // Error detected
   GetTaskCycle_Diag_Memo:=GetTaskCycle_Diag;
   GetTaskCycle_Err:=TRUE;
   ELSE // Valid Task interval
   TaskInterval_Memo:=TaskInterval;
   GetTaskCycle_Err:=FALSE;
END_IF;
```

SetCurrentTaskCycle: Sets the Cyclic Task Interval

Function Description

This function sets the Cyclic Task interval with the input parameter value in microseconds (µs) and returns an operation diagnostic. If the operation has been executed successfully, the new interval is valid at the start of the next iteration of the Cyclic Task (no effect on the current execution).

NOTE: The value passed to the function must be a multiple of a thousand. If not, an error code will be returned.

If you define an interval for a Cyclic Task that is too short (shorter than the effective duration of the task and any other processing required by the application), it will repeat immediately after the write of the outputs without executing other lower priority tasks or any system processing. This will affect the execution of all tasks and cause the controller to exceed the system watchdog limits, generating a system watchdog exception and stopping your controller.

NOTICE
UNINTENDED INTERRUPTION OF PROGRAM EXECUTION Test your new Cyclic Task interval time to validate that is appropriate to prevent a system watchdog exception before calling the `SetCurrentTaskCycle` function. **Failure to follow these instructions can result in equipment damage.**

NOTE: You must test the possible range of values prior to commissioning your application, and be sure your code validates the new interval to those test conditions.

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 13)*.

I/O Variables Description

The following table describes the input variable:

Input	Type	Description
`newTime`	`UDINT`	New value of the POU attached Task interval in microseconds (µs). **NOTE:** If valid, the new value is operational at next cycle.

The following table describes the output variable:

Output	Type	Description
`SetCurrentTaskCycle`	`UDINT`	Function operation diagnostic: 0 = No error detected 1 = General (internal) error detected 2 = Invalid parameter detected (out of range value) 12 = Function not implemented in the controller 24 = Functionality not supported (for example non Cyclic Task)

NOTE: This function changes the current Cyclic Task interval. It does not override the Cyclic Task interval parameter value set in the task configuration. Initial configuration parameters are restored on Reset, Reboot or Download command.
Refer to your controller programming guide for more information about your controller states and behaviors.

NOTE: Do not use the SetCurrentTaskCycle function in the same task defined as the CANopen managers bus cycle task. Changing the cycle time of this task will affect the heartbeat or the nodeguarding, possibly causing the CANopen devices to detect an error and pass into a fallback state.

⚠ CAUTION

UNINTENDED INTERRUPTION OF PROGRAM EXECUTION

Do not use the `SetCurrentTaskCycle` function in any task defined as a bus cycle task.

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

Example

The following example describes how to set the current interval of the Cyclic Task attached to a POU `SetTaskInterval` in CFC, FBD or LD language.

The function `SetCurrentTaskCycle` is executed when the flag `SetTaskInterval_Flag` is TRUE (this flag is automatically reset to FALSE after execution) and the function operation diagnostic is stored in the `SetTaskInterval_Diag` variable.

NOTE:
The function enabling input/output (EN/ENO) is activated:

- In CFC: right click on the function and select the **EN/ENO** command.
- In FBD: add the function with the command **Insert Box with EN/ENO** of the **FBD/LD/IL** menu.
- In LD: switch to FBD view with the **View** command of the **FBD/LD/IL** menu, insert the function as described above and switch back to LD view.

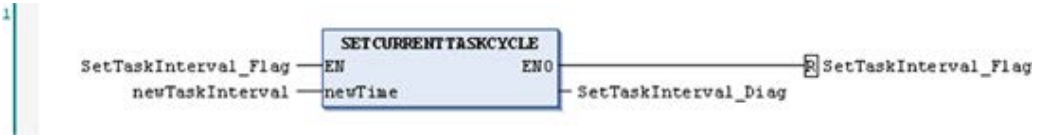
Variables declaration:

```
PROGRAM SetTaskInterval
VAR
   // Set a new Task Interval command flag
   SetTaskInterval_Flag: BOOL := FALSE;
   // Value of the new Cyclic Task interval value (µs)
   newTaskInterval: UDINT;
   //SetCurrentTaskCycle function operation diagnostic
   SetTaskInterval_Diag: UDINT := 0;
END_VAR
```

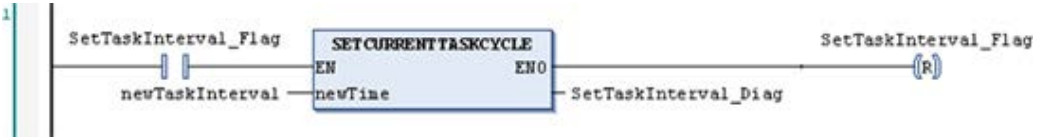
Program in CFC language:



Program in FBD language:



Program in LD language:



Appendices

Appendix A
Function and Function Block Representation

Overview

Each function can be represented in the following languages:

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

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

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Differences Between a Function and a Function Block	14
How to Use a Function or a Function Block in IL Language	15
How to Use a Function or a Function Block in ST Language	18

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.

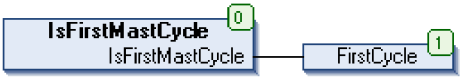
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

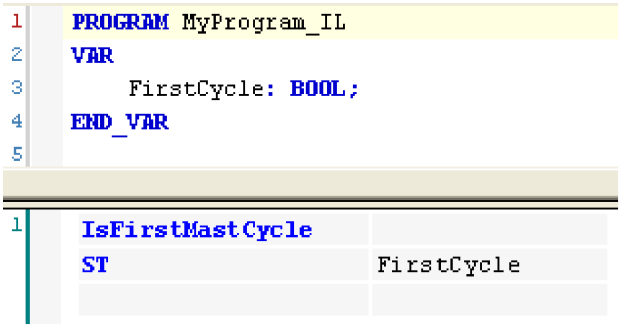
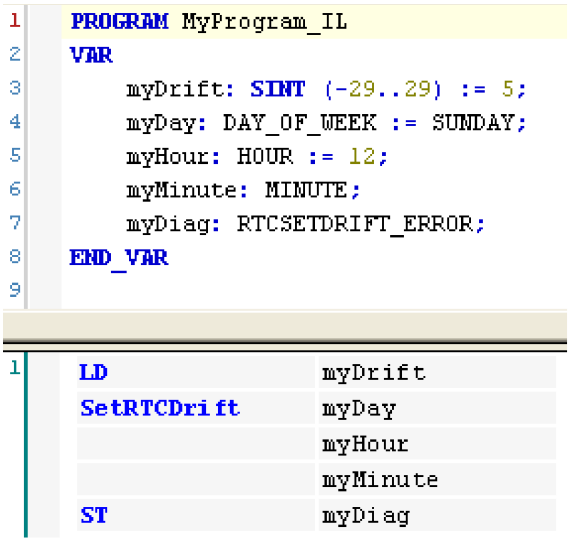
This procedure describes how to insert a function in IL language:

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 EcoStruxure Machine Expert, 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`	
IL example of a function with input parameters: `SetRTCDrift`	

Using a Function Block in IL Language

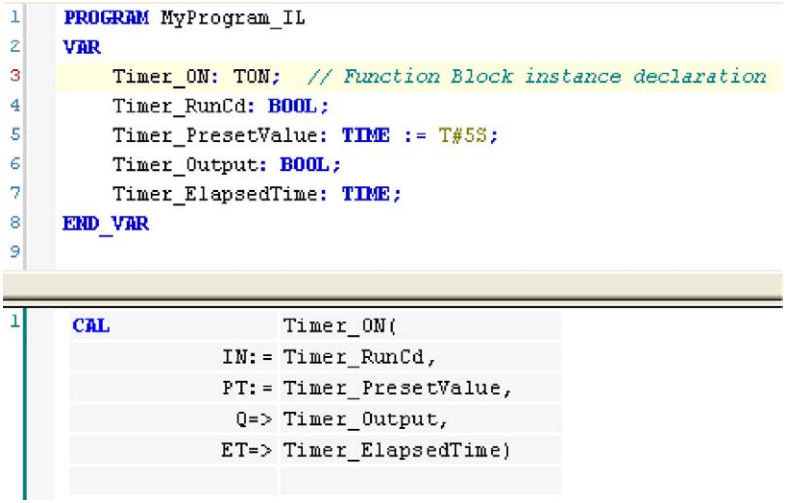
This procedure describes how to insert a function block in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see EcoStruxure Machine Expert, 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 EcoStruxure Machine Expert, 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

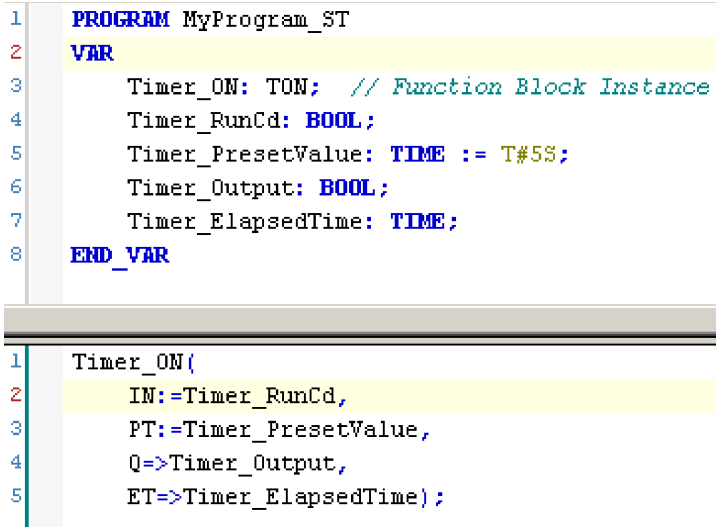
This procedure describes how to insert a function block in ST language:

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 EcoStruxure Machine Expert, 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);
```

Glossary

B

byte

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

C

CFC

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

F

FB

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

function block diagram

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

I

IL

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

INT

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

L

LD

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

P

POU

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

S

ST

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

V

variable

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

Index

F

functions
- differences between a function and a function block, *14*
- how to use a function or a function block in IL language, *15*
- how to use a function or a function block in ST language, *19*

G

GetCurrentTaskCycle
- Toolbox_Advance, *14*

S

SetCurrentTaskCycle
- Toolbox_Advance, *16*

T

Toolbox_Advance
- GetCurrentTaskCycle, *14*
- SetCurrentTaskCycle, *16*

EcoStruxure Machine Expert
Twido Emulation Support Library

Library Guide

06/2019

EIO0000002956.00

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2019 Schneider Electric. All rights reserved.

Table of Contents

Safety Information 5
About the Book 9
Chapter 1 General Information on the TwidoEmulationSupport Library 13
General Information on the TwidoEmulationSupport Library 13
Chapter 2 Standard Function Blocks 15
FB_Timer: Timer 16
FB_Counter: Counter 18
FB_FiFo and FB_LiFo: Register Function Block 19
FB_ShiftBitRegister: Shift Bit Register 20
Chapter 3 Advanced Function Blocks 21
FB_ScheduleBlock: Schedule Block 22
FB_Drum: Drum Controller 25
FB_StepCounter: Step Counter 28
FB_PID: PID Function Block 29
Chapter 4 Communication Function Blocks 33
FB_EXCH: Exchange Function Block 34
FB_ReadVar: Read Data Function Block 37
FB_WriteVar: Write Data Function Block 39
FB_WriteReadVar: Write and Read Function Block 41
FB_SendRecvMsg: Send and Receive Message Function Block 43
Chapter 5 Twido Functions 45
Twido Functions 45
Glossary 63
Index 65

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.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

BEFORE YOU BEGIN

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

⚠ WARNING

UNGUARDED EQUIPMENT

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

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

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

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

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

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

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

START-UP AND TEST

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

⚠ WARNING

EQUIPMENT OPERATION HAZARD

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

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

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

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

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

Before energizing equipment:

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

OPERATION AND ADJUSTMENTS

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

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

About the Book

At a Glance

Document Scope

This document describes the TwidoEmulationSupport library, which provides the functions and function blocks to convert projects from EcoStruxure Machine Expert - Basic / TwidoSoft / TwidoSuite into a compatible implementation in EcoStruxure Machine Expert. For more information on Twido, refer to Converting EcoStruxure Machine Expert - Basic and Twido Projects *(see EcoStruxure Machine Expert, Programming Guide)* and to the *Controller Feature Sets for Migration* lists *(see EcoStruxure Machine Expert, Programming Guide)*.

Validity Note

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

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

Related Documents

Title of Documentation	Reference Number
EcoStruxure Machine Expert Programming Guide	*EIO0000002854 (ENG)*; *EIO0000002855 (FRE)*; *EIO0000002856 (GER)*; *EIO0000002857 (ITA)*; *EIO0000002858 (SPA)*; *EIO0000002859 (CHS)*
EcoStruxure Machine Expert - Basic Generic Functions Library Guide	*EIO0000003289 (ENG)*; *EIO0000003290 (FRE)*; *EIO0000003291 (GER)*; *EIO0000003292 (SPA)*; *EIO0000003293 (ITA)*; *EIO0000003294 (CHS)*; *EIO0000003295 (POR)*; *EIO0000003296 (TUR)*

Title of Documentation	Reference Number
EcoStruxure Machine Expert - Basic Operating Guide	EIO0000003281 (ENG); EIO0000003282 (FRE); EIO0000003283 (GER); EIO0000003284 (SPA); EIO0000003285 (ITA); EIO0000003286 (CHS); EIO0000003287 (POR); EIO0000003288 (TUR)
EcoStruxure Machine Expert Modbus and ASCII Read/Write Functions PLCCommunication Library Guide	EIO0000002962 (ENG); EIO0000002963 (FRE); EIO0000002964 (GER); EIO0000002965 (SPA); EIO0000002966 (ITA); EIO0000002967 (CHS)

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

Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.

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

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

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

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

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

Chapter 1
General Information on the TwidoEmulationSupport Library

General Information on the TwidoEmulationSupport Library

Conversion of Function Blocks

The following options are provided for the conversion of function blocks:

- The function block is converted to a type that already exists in EcoStruxure Machine Expert, for example the function block exists in the standard library.
  For more information on standard libraries, refer to the online help *CoDeSys Libraries/ Standard Library*.
- A new function block that is fully compatible with EcoStruxure Machine Expert - Basic is implemented and part of the TwidoEmulationSupport library.
- For functions and function blocks that are not intended to be used in a EcoStruxure Machine Expert project, the converter can generate a type with no implementation.

For more information on the conversion of function blocks, refer to Converting EcoStruxure Machine Expert - Basic and Twido Projects *(see EcoStruxure Machine Expert, Programming Guide)*.

Chapter 2
Standard Function Blocks

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
`FB_Timer`: Timer	16
`FB_Counter`: Counter	18
`FB_FiFo` and `FB_LiFo`: Register Function Block	19
`FB_ShiftBitRegister`: Shift Bit Register	20

FB_Timer: Timer

Overview

The following graphic shows the pin diagram of the function block FB_Timer:



The function block %TM in EcoStruxure Machine Expert - Basic can configure 3 types of timers:

- TON (Timer On-Delay): this type of timer is used to control on-delay actions.
- TOF (Timer Off-Delay): this type of timer is used to control off-delay actions.
- TP (Timer-Pulse): this type of timer is used to create a pulse of a precise duration.

For more information refer to the CoDeSys online help in EcoStruxure Machine Expert: *CoDesSys-Libraries/Standard Library/Timer*.

NOTE: When changing the timer type online while i_xEnable is TRUE, the formerly selected timer type is reset and the new timer type starts. FB outputs behave according the newly selected timer.

I/O Variables Description

The TwidoEmulationSupport library provides a function FB_Timer, which calls the timer type TON, TOF or TP depending on the input i_etTimerType. The given TimeBase is used for calculating the preset time.

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data type	Description
i_xEnable	BOOL	Rising edge starts counting up q_iCurrentValue.
i_iPreset	INT	Initial value of the preset value (0..32767)
i_etTimerType	ET_TimerType	Indicates the timer type: TON, TOF, or TP
i_etTimeBase	ET_TimerTimeBases	Time base of 1 ms, 10 ms, 100 ms, 1 s, 1 min Default: 1 min
i_etMaxVal	ET_TimerCounterMaxVal	Maximum value

Input	Data type	Description
i_xRetentive	BOOL	Behavior of the timer: ● TRUE: timer keeps value of the output q_iCurrentValue when disabled, and continues counting from previous value when enabled again ● FALSE: timer resets the output q_iCurrentValue after disable

The data type for the input variable i_etTimerType is:

```
TYPE ET_TimeType:
(
TON    := 1,   //   On-delay Timer
TOF    := 2,   //   Off-delay Timer
TP     := 3,   //   Pulse Timer
) INT;
END_TYPE
```

NOTE: If an invalid value is given to i_etTimerType, then q_xOutput stays at FALSE and q_iCurrentValue stays at 0.

The data type for the input variable i_etTimeBase is:

```
TYPE ET_TimerTimeBases:
(
tb1MS     := 1,   //   1 millisecond time base
tb10MS    := 2,   //   10 millisecond time base
tb100MS   := 3,   //   100 millisecond time base
tb1S      := 4,   //   10 second time base
tb1M      := 5,   //   1 minute time base
) INT;
ET_TYPE
```

The data type for the input variable i_etMaxVal is:

```
TYPE ET_TimerCounterMaxVal:
(
Val9999   :=  0,  //    Value  9999
Val32767  :=  1   //    Value  3267
) INT
END_TYPE
```

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data type	Description
q_xOutput	BOOL	Timer output
q_iCurrentValue	INT	Elapsed time

FB_Counter: Counter

Overview

The function block FB_Counter provides up and down counting of events.

The following graphic shows the pin diagram of the function block FB_Counter:



I/O Variables Description

The TwidoEmulationSupport library provides a function block FB_Counter which calls the function block CTUD. The standard CTUD behavior is extended by a preset value handling and Twido-compliant overflow behavior.

For more information on counters refer to the CoDeSys online help in EcoStruxure Machine Expert *Standard Library / Counter*.

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
i_xLoad	BOOL	Sets the current value to the preset value.
i_xReset	BOOL	Resets the input.
i_xCountUp	BOOL	Counts up the input.
i_xCountDown	BOOL	Counts down the input.
i_iPreset	INT	Preset value Default: 32767
i_etMaxVal	ET_TimerCounterMaxVal	Counter and preset maximum value

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data Type	Description
q_xUnderflow	BOOL	Counts down underflow / empty
q_xDone	BOOL	q_iCounterValue = i_iPreset / preset reached / done
q_iCounterValue	INT	Current value (0..32767)
q_xOverflow	BOOL	Up-count overflow full

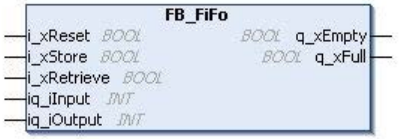
FB_FiFo and FB_LiFo: Register Function Block

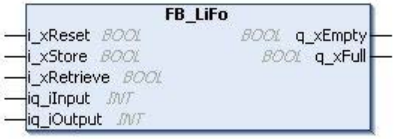
Overview

The register function block in EcoStruxure Machine Expert - Basic has 2 types:

- Queue (FIFO)
- Stack (LIFO)

The following graphics show the pin diagrams of the function blocks FB_FiFo and FB_LiFo:





I/O Variables Description

The table describes the input / output variables of the function block in the TwidoEmulationSupport library:

Input / Output	Data Type	Description
iq_iInput	INT	Input word (read and write variable)
iq_iOutput	INT	Output word (read and write variable)

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
i_xReset	BOOL	Reset input initializes the register at state = 1
i_xStorage	BOOL	The storage input stores the contents in the register, on a rising edge.
i_xRetrieve	BOOL	The retrieval input loads a data word of the register, on a rising edge.

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data Type	Description
q_xEmpty	BOOL	This output indicates that the register is empty.
q_xFull	BOOL	This output indicates that the register is full.

FB_ShiftBitRegister: Shift Bit Register

Overview

The function block FB_ShiftBitRegister provides a left or right shift of binary data bits. This is not a rotating bit shift. Overflow bits are lost.

The following graphic shows the pin diagram of the function block FB_ShiftBitRegister:



I/O Variables Description

The table describes the input / output variables of the function block in the TwidoEmulationSupport library:

Input / Output	Data Type	Description
iq_iRegister (16 Bit)	INT	Register bit

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
i_xReset	BOOL	The reset input resets all bits to 0.
i_xShiftLeft	BOOL	On a rising edge, shifts a register bit to the left.
i_xShiftRight	BOOL	On a rising edge, shifts a register bit to the right.

Chapter 3
Advanced Function Blocks

What Is in This Chapter?

This chapter contains the following topics:

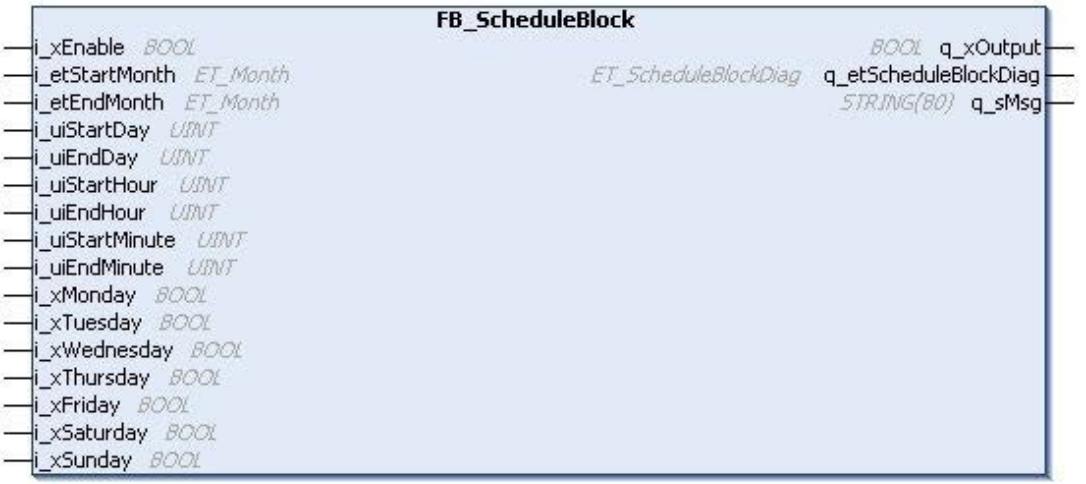
Topic	Page
`FB_ScheduleBlock`: Schedule Block	22
`FB_Drum`: Drum Controller	25
`FB_StepCounter`: Step Counter	28
`FB_PID`: PID Function Block	29

FB_ScheduleBlock: Schedule Block

Overview

The function block FB_ScheduleBlock is used to control actions at a defined month, day, and time.

The following graphic shows the pin diagram of the function block FB_ScheduleBlock:



I/O Variables Description

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
i_xEnable	BOOL	Enable the function block FB_ScheduleBlock.
i_etStartMonth	ET_Month	The month to set the output. (January to December).
i_etEndMonth	ET_Month	The month to reset the output. (January to December).
i_uiStartDay	UINT	[1..31] Activation start day
i_uiEndDay	UINT	[1..31] Activation end day
i_uiStartHour	UINT	[0..23] Activation start hour
i_uiEndHour	UINT	[0..23] Activation end hour
i_uiStartMinute	UINT	[0..59] Activation start minute
i_uiEndMinute	UINT	[0..59] Activation end minute
i_xMonday	BOOL	Run activity on Monday

Input	Data Type	Description
i_xTuesday	BOOL	Run activity on Tuesday
i_xWednesday	BOOL	Run activity on Wednesday
i_xThursday	BOOL	Run activity on Thursday
i_xFriday	BOOL	Run activity on Friday
i_xSaturday	BOOL	Run activity on Saturday
i_xSunday	BOOL	Run activity on Sunday

```
TYPE ET_Month:
(
January        :=     1,
…
December       :=     12
) INT;
END_TYPE
```

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data Type	Description
q_xOutput	BOOL	This output is set to 1 when the present date and time are between the setting of the start of the active period and the setting of the end of the active period.
q_etScheduleBlockDiag	ET_ScheduleBlockDiag	Diagnostic code ET_ScheduleBlockDiag.
q_sMsg	String	Diagnostic message

```
TYPE ET_ScheduleBlockDiag :
(
        // TES errors
        OK                          := 0,
        InputStartDayInvalid        := 1,
        InputEndDayInvalid          := 2,
        InputStartHourInvalid       := 3,
        InputEndHourInvalid         := 4,
        InputStartMinuteInvalid     := 5,
        InputEndMinuteInvalid       := 6,

        // DTU errors
        FirstError                  := 5750,
        TimeOut                     := 5751,
        NotAvailable                := 5752,
        InputInvalid                := 5753,
        FirstMF                     := 5770,
        LastError                   := 5799,

        Unknown                     := 5800
) INT;
END_TYPE
```

The data structure DTU.ERROR (ENUM) describes errors which occur when using the functions of the CAA_DtUtility library. Error code range 5750-5799 is reserved for this library in the prefix registration for libraries. For more information, refer to *CAA libraries/CAA_DTUtil.library/Data types/Enumerations/DTU.ERROR (ENUM)*.

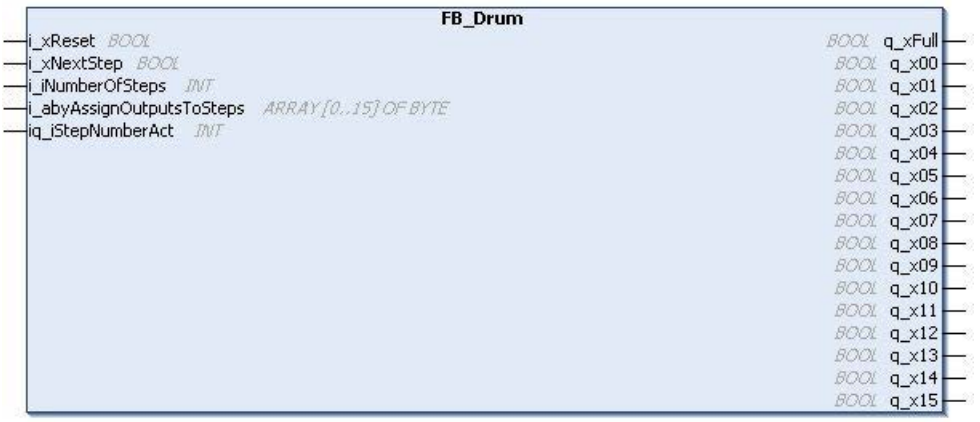
FB_Drum: Drum Controller

Overview

The drum controller operates on a principle similar to an electromechanical drum controller. The drum can provide up to 8 states which are engaged cyclically. While a rising edge at the input `i_xNextStep` turns the drum further, the actual step number can also be set by the software.

Each drum state activates a pattern of up to 16 control bits so that the drum controller represents a kind of state machine.

The following graphic shows the pin diagram of the function block `FB_Drum`:



I/O Variables Description

The table describes the input / output variables of the function block in the TwidoEmulationSupport library:

Input / Output	Data Type	Description
`iq_iStepNumberAct`	INT	Current step number which can be read and written. When written, the effect takes place on the next execution of the function block.

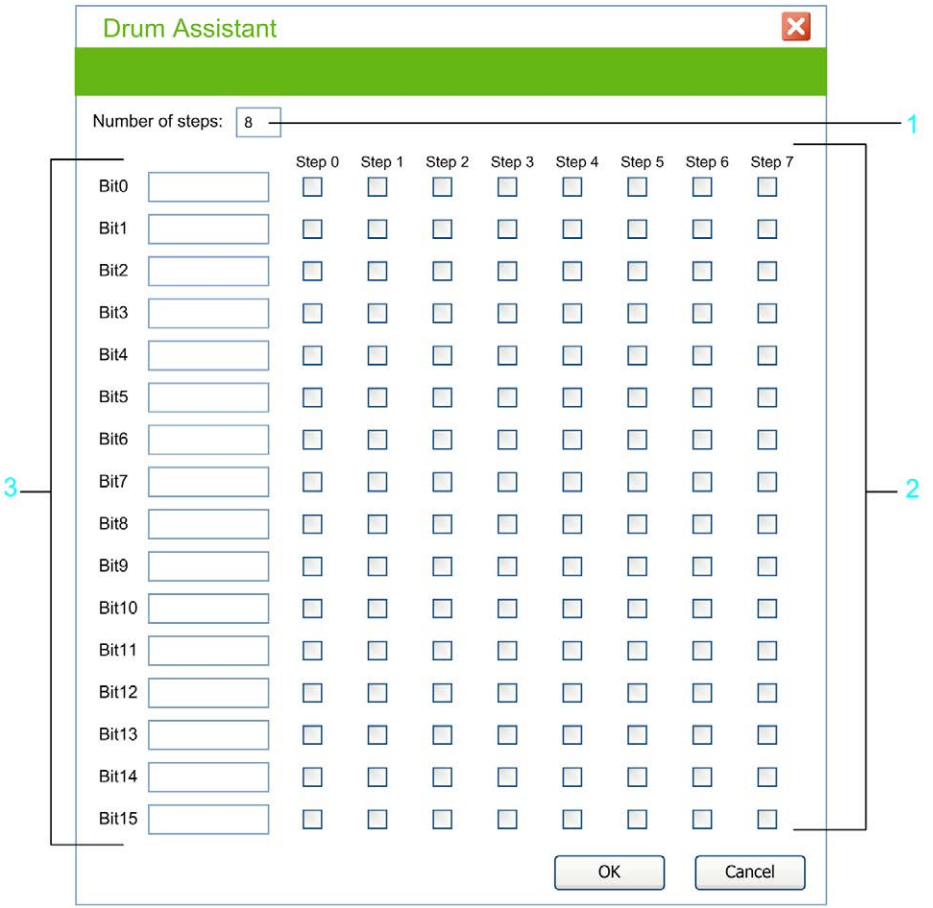
The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
`i_xReset`	BOOL	The Reset input sets the drum controller to step 0.
`i_xNextStep`	BOOL	A rising edge at this input causes the drum controller to advance by one step and updates the control bits.
`i_iNumberOfSteps`	INT	1-8 (number of steps)
`i_abyAssignOutputsToSteps`	ARRAY OF BYTE	Assignment of outputs 0..15 to steps 0..7

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data Type	Description
q_xFull	BOOL	Full output indicates that the current step equals the last step defined.
q_x00 - q_x15	BOOL	Outputs or internal bits associated with the step (16 control bits) and defined in the EcoStruxure Machine Expert - Basic Configuration Editor.

The configuration of FB_Drum is not a configuration but an entry array of the function block. The following graphic shows the **Drum Assistant** in EcoStruxure Machine Expert - Basic:



1 Number of steps available in the drum controller (up to 8).
2 A 8x16 bitmask to assign states to all bit outputs (ARRAY [0..15] OF BYTE represents this bitmask).
3 16 bit outputs

New function block in the TwidoEmulationSupport library:



FB_StepCounter: Step Counter

Overview

The function block FB_StepCounter provides a series of steps to which actions can be assigned. Moving from one step to another depends on external or internal events.

The following graphic shows the pin diagram of the function block FB_StepCounter:



I/O Variables Description

The table describes the input / output variables of the function block in the TwidoEmulationSupport library:

Input / Output	Data Type	Description
iq_axCurrentStep	ARRAY OF BOOL	Step counter bits 0 to 255.

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
i_xReset	BOOL	Resets the step counter.
i_xUp	BOOL	Shift to left input increments the step counter by one step, on a rising edge.
i_xDown	BOOL	Shift to right input decrements the step counter by one step, on a rising edge.

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data Type	Description
q_byCurrentStepNumber	BYTE	Step counter bits 0 to 255.

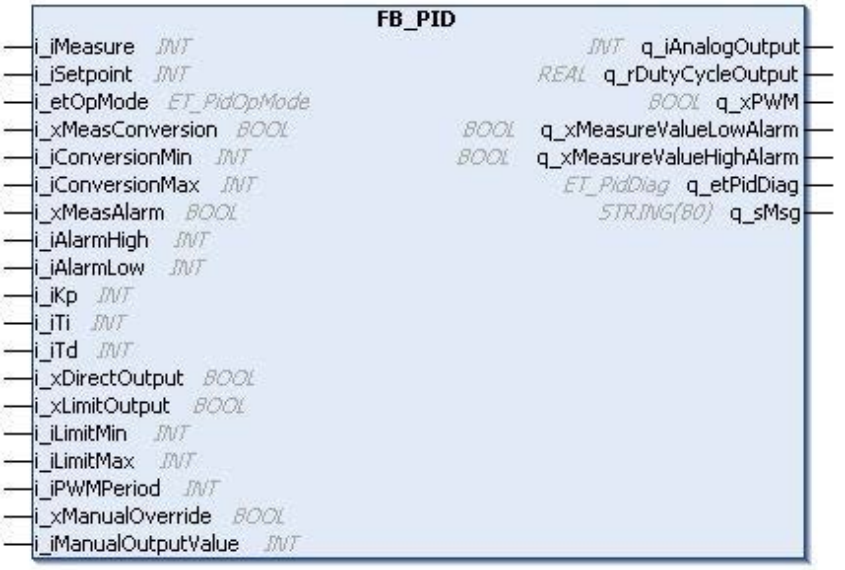
Each time a step is active, the associated bit is set to 1. The step counter can control output bits (memory bits). Only one step of a step counter can be active at a time. If one bit in iq_axCurrentStep is set externally, all other bits are reset and q_byCurrentStepNumber is set accordingly.

FB_PID: PID Function Block

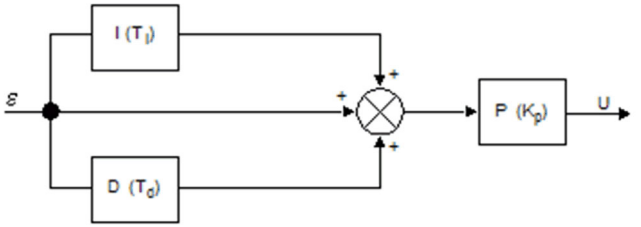
Overview

The function block FB_PID provides a PID controller.

The following graphic shows a pin diagram of the function block FB_PID:



The main algorithm is represented by the following flow chart:



The main algorithm is calculated as:

$$u(i) = K_P * \left\{ s(i) + \frac{T_S}{T_I} \sum_{j=1}^{i} s(j) + \frac{T_d}{T_S} [s(i) - s(i-1)] \right\}$$

While a derivative time constant of Td=0 will disable the derivative branch of the PID controller, a proportional gain Kp of 0 is prohibited. This would result in a gain of 100%.

Setting the integral time constant Ti=0 will switch to an alternate calculation rule:

$$u(i) = K_P * \left\{ s(i) + \frac{T_d}{T_S}[s(i) - s(i-1)) \right\} + 5000$$

This would center the analog output signal which is in a range [0..10000].

The measured value can be converted into a parametrized range. This new range will then be applied to the setpoint as well as to the measure alarm levels.

In each case, the output is in a range [0..10000] but can be limited.

I/O Variables Description

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
i_iMeasure	INT	Control variable [1..10000]
i_iSetpoint	INT	Controller setpoint [1..10000] or [configured min..configured max]
i_etOpMode	ET_PidOpMode	PID / PI - in case of PID it can be overwritten by etCorrectorType for compatibility purposes.
i_xMeasConversion	BOOL	Activates the conversion of measured values to the given bounds from [0..10000] to [i_iConversionMin..i_iConversionMax].
i_iConversionMin	INT	Conversion of the minimum value.
i_iConversionMax	INT	Conversion of the maximum value.
i_xMeasAlarm	BOOL	Activates the measured value range alarms.
i_iAlarmHigh	INT	High alarm threshold value for q_iAnalogOutput.
i_iAlarmLow	INT	Low alarm threshold value for q_iAnalogOutput.
i_iKp	INT	Proportional gain factor
i_iTi	INT	Integral time constant
i_iTd	INT	Derivative time constant (is ignored in case of PI)
i_xDirectOutput	BOOL	TRUE: direct action FALSE: reverse action
i_xLimitOutput	BOOL	Activates the output limitation.
i_iLimitMin	INT	Output lower limit
i_iLimitMax	INT	Output upper limit
i_iPWMPeriod	INT	PWM signal

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data Type	Description
q_iAnalogOutput	INT	Controller output to be digital to analog converted [1..10000]
q_rDutyCycleOutput	REAL	Reduced to a value [0..1] which can be assigned to a PWM module manually.
q_xPWM	BOOL	Software generated PWM output
q_xMeasureValueLowAlarm	BOOL	Monitoring of the control variable. Is set to 1, if the low alarm level is underrun.
q_xMeasureValueHighAlarm	BOOL	Monitoring of the control variable. Is set to 1, if the high alarm level is exceeded.
q_etPidDiag	ET_PidDiag	Diagnostic code from the PID controller
q_sMsg	STRING	Diagnostic message

The PID block configuration in EcoStruxure Machine Expert - Basic provides the additional parameter sampling period (in the **PID** tab of the **PID Assistant**). The sampling period is the cycle time of the task which executes the PID block. To help ensure that the signal at q_xPWM is generated properly, the configured i_iPWMPeriod must be a multiple of the sample time. A PWM period of 0 deactivates the signal generation.

The i_etOpMode input is of the type:

```
TYPE ET_PidOpMode :
(
       PID            := 0,
       // AutoPID     := 1,          // not supported by TES
       // Autotune    := 2,          // not supported by TES
       PI             := 4
) INT;
END_TYPE
```

The `i_etPidDiag` output is of the type:

```
TYPE ET_PidDiag :
(
        NotActive                       := 16#0000,
        InProgress                      := 16#2000,
        SetpointReached                 := 16#4000,
        // AutotunePhase1               := 16#0100,          // Autotune not supported by TES.FB_PID
        // AutotunePhase2               := 16#0200,          // Autotune not supported by TES.FB_PID
        // AutotunePhase3               := 16#0400,          // Autotune not supported by TES.FB_PID
        // AutotunePhase4               := 16#0800,          // Autotune not supported by TES.FB_PID
        // AutotuneComplete             := 16#1000,          // Autotune not supported by TES.FB_PID
        OpModeRangeExceeded             := 16#8001,
        ConversionLimitsEqual           := 16#8002,
        ConversionLimitsSwapped         := 16#8003,
        OutputLimitOutOfConvRange       := 16#8004,
        OutputLimitOutOfRange           := 16#8005,
        SetpointOutOfConvRange          := 16#8006,
        SetpointOutOfRange              := 16#8007,
        // ATstartWrongBehaviour        := 16#8008           // Autotune not supported by TES.FB_PID
        // Autotuning errors not listed

        // Additional error states in TES.FB_PID
        AlarmValueExceedsRange          := 16#80B1,
        ControlParamExceedsRange        := 16#80B2,
        SamplePeriodExceedsRange        := 16#80B3,
        ManualValueExceedsRange         := 16#80B4
) INT;
END_TYPE
```

Chapter 4
Communication Function Blocks

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
FB_EXCH: Exchange Function Block	34
FB_ReadVar: Read Data Function Block	37
FB_WriteVar: Write Data Function Block	39
FB_WriteReadVar: Write and Read Function Block	41
FB_SendRecvMsg: Send and Receive Message Function Block	43

FB_EXCH: Exchange Function Block

Overview

The following graphic shows the pin diagram of the function block FB_EXCH:



The M221 controller can communicate with a Modbus slave device or can send/receive messages in character mode (ASCII).

Twido and EcoStruxure Machine Expert - Basic provide the following functions for communication:

- EXCH instruction to transmit/receive messages
- Exchange control function block (MSG) to control the data exchanges

The TwidoEmulationSupport library handles the communication with the function block FB_EXCH. This function block uses the function block SEN.SEND_RECV_MSG of the PLCCommunication library. This has the functionality to send and receive user-defined messages and waits for a response.

I/O Variables Description

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data Type	Description
i_xExecute	BOOL	The functionality starts on rising edge.
i_xReset	BOOL	The current message transmission stops on rising edge and the communication reinitializes.
i_byPort	BYTE	[1..3] communication port 1 = Serial port 1 2 = Serial port 2 3 = Ethernet

Input	Data Type	Description
i_pbyBuffer	POINTER TO BYTE	Pointer to send and/or receive buffer. The first 2 words are control words. First 2 words: 4 control bytes: 1. Control byte length: The length byte contains the length of the transmission table in bytes (250 maximum), which is overwritten by the number of characters received at the end of the reception (if reception is requested). 2. Control byte command: [0..2] 0 = Transmission only 1 = Send/receive 2 = Reception only 3. Control byte SndBytePosDiscard: Byte position is not sent. 4. Control byte RcvBytePosDiscard: Received byte position is discarded.
i_uiLengthInByte	UINT	Length of send/receive buffer + 2 control words
i_xAsciiMode	BOOL	TRUE = ASCII mode configured. FALSE = Modbus TCP configured.

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data Type	Description
q_xBusy	BOOL	q_xBusy is set to TRUE while the function is ongoing.
q_xDone	BOOL	q_xDone is set to TRUE when the function is completed successfully.
q_xError	BOOL	q_xError is set to TRUE when the function stops because of a detected error.
q_etExchDiag	ET_ExchDiag	Diagnostic code.
q_sMsg	STRING [80]	Diagnostic message.

The function block FB_EXCH has the following error codes:

```
TYPE ET_ExchDiag :
(

    OK                              := 0,   // operation was successful
    LengthToGreat                   := 1,   // number of bytes to be transmitted is too great (> 128)
    SndTableToSmall                 := 2,   // transmission table too small
    BufferToSmall                   := 3,   // word table too small
    RcvTableOverflow                := 4,   // receive table overflowed
    TransmissionError               := 6,   // transmission
    BadCommand                      := 7,   // incorrect command within table
    PortNotConfiguredOrNotAvailable := 8,   // selected port not configured/available
    ReceptionError                  := 9,   // detected reception error
    SndBytePosDiscardToGreat        := 11,  // transmission offset larger than transmission table
    RcvBytePosDiscardToGreat        := 12,  // reception offset larger than reception table
    NoSuchIpAddress                 := 101,// no such IP address
    TcpConnBroken                   := 102,// the TCP connection is broken
    RejectionOnConnectionAttempt    := 110,// rejection on connection attempt
    HardwareOrConfigError           := 121,// detected hardware or configuration error
                                           (MAC, Chip, Duplicated IP)
    ReceivingTimeOutElapsed         := 122,// receiving timed-out elapsed after data was sent
    BufferToGreat                   := 201, // Additional errors are not defined in Twido
    PortNumberInvalid               := 202, // Additional errors are not defined in Twido
    SenAddmError                    := 203, // Additional errors are not defined in Twido
    SenSendRecvMsgCommError         := 204, // Additional errors are not defined in Twido
    SenSendRecvMsgOperError         := 205, // Additional errors are not defined in Twido
    CommResetByUserRequest          := 206, // Additional errors are not defined in Twido
    NoMulticastPerTCP               := 207  // Additional errors are not defined in Twido

) INT;
END_TYPE
```

FB_ReadVar: Read Data Function Block

Overview

The following graphic shows the pin diagram of the function block FB_ReadVar:



The function block FB_ReadVar reads data from an external device using the Modbus SL or Modbus TCP protocol.

I/O Variables Description

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data type	Description
i_xExecute	BOOL	The function is executed on the rising edge of this input. **NOTE:** When ixExecute is set to TRUE at the first task cycle in RUNNING after a cold or warm reset, the rising edge is not detected.
i_xAbort	BOOL	Stops the ongoing operation at the rising edge.
i_iLink	INT	Indicates the port used to communicate. ● 1 = Serial port 1 ● 2 = Serial port 2 ● 3 = Ethernet / Modbus TCP
i_iId	INT	Device identifier of targeted external device: ● Serial line slave address: 1...247 or ● Ethernet index: 1...16 (Depends on the link configuration).
i_iTimeout	INT	Set the maximum time to wait to receive a response from the modem. Specified in ms. A value of 0 means no timeout is enforced.

Input	Data type	Description
i_etObjType	SEN.ObjectType	Value of enumerated data type (SEN.ObjectType *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*) contains the object types that are available for writing: • MW: Read holding register (16 bit) • I: Read digital input (1 bit) • Q: Read coils (1 bit) • IW: Read input register (16 bits)
i_diFirstObj	DINT	Address of the first object from which values are read.
i_iQuantity	INT	Number of objects to read.
i_pbyBuffer	POINTER TO BYTE	Address of the buffer in which read object values are stored.
i_astRemoteDevices	ARRAY [1..Gc_uiRemoteAddressTableLen] OF ST_RemoteDevice	Contains the list of remote device addresses for Modbus TCP communication.

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data type	Description
q_xDone	BOOL	q_xDone is set to TRUE when the function is completed successfully.
q_xBusy	BOOL	q_xBusy is set to TRUE while the function is ongoing.
q_xAborted	BOOL	q_xAborted is set to TRUE when the function is stopped with the i_xAbort input.
q_xError	BOOL	q_xError is set to TRUE when the function stops because of a detected error.
q_etCommError	SEN.CommunicationErrorCodes	q_etCommError contains the value of communication error codes SEN.CommunicationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.
q_etOperError	SEN.OperationErrorCodes	q_etOperError contains the value of operation error codes SEN.OperationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.

FB_WriteVar: Write Data Function Block

Overview

The following graphic shows the pin diagram of the function block FB_WriteVar:



The function block FB_WriteVar writes data to an external device using the Modbus SL or Modbus TCP protocol.

I/O Variables Description

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data type	Description
i_xExecute	BOOL	The function is executed on the rising edge of this input. **NOTE:** When xExecute is set to TRUE at the first task cycle in RUNNING after a cold or warm reset, the rising edge is not detected.
i_xAbort	BOOL	Stops the ongoing operation at the rising edge.
i_iLink	INT	Indicates the port used to communicate. ● 1 = Serial port 1 ● 2 = Serial port 2 ● 3 = Ethernet / Modbus TCP
i_iId	INT	Device identifier of targeted external device: ● Serial line slave address: 1...247 or ● Ethernet index: 1...16 (Depends on the link configuration). **NOTE:** Value 0 is an invalid value and does not represent a broadcast message.
i_iTimeout	INT	Set the maximum time to wait to receive a response. Specified in ms. A value of 0 means no timeout is enforced.
i_etObjType	SEN.ObjectType	Value of enumerated data type (SEN.ObjectType *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*) contains the object types that are available for writing: ● MW: Write multiple registers (16 bits) ● Q: Write multiple coils (1 bit)

Input	Data type	Description
i_diFirstObj	DINT	Address of the first object from which values are written.
i_iQuantity	INT	Number of objects to write.
i_pbyBuffer	POINTER TO BYTE	Address of the buffer in which object values to write are stored.
i_astRemoteDevices	ARRAY [1..Gc_uiRemoteAddressTableLen] OF ST_RemoteDevice	Contains the list of remote device addresses for Modbus TCP communication.

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data type	Description
q_xDone	BOOL	q_xDone is set to TRUE when the function is completed successfully.
q_xBusy	BOOL	q_xBusy is set to TRUE while the function is ongoing.
q_xAborted	BOOL	q_xAborted is set to TRUE when the function is stopped with the i_xAbort input.
q_xError	BOOL	q_xError is set to TRUE when the function stops because of a detected error.
q_etCommError	SEN.CommunicationErrorCodes	q_etCommError contains the value of communication error codes SEN.CommunicationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.
q_etOperError	SEN.OperationErrorCodes	q_etOperError contains the value of operation error codes SEN.OperationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.

FB_WriteReadVar: Write and Read Function Block

Overview

The function block FB_WriteReadVar can be used to:

- write consecutive internal registers and read back their values in the same execution cycle of the function block (single transaction),
- write consecutive internal registers and read different registers.

The following graphic shows the pin diagram of the function block FB_WriteReadVar:



The function block FB_WriteReadVar reads and writes internal registers (MW type only) to an external device in the Modbus protocol. The read and write operations are contained in a single transaction. Note that the write operation is performed first.

I/O Variables Description

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data type	Description
i_xExecute	BOOL	The function is executed on the rising edge of this input. **NOTE:** When i_xExecute is set to TRUE at the first task cycle in RUNNING after a cold or warm reset, the rising edge is not detected.
i_xAbort	BOOL	Stops the ongoing operation at the rising edge.
i_iLink	INT	Indicates the port used to communicate. ● 1 = Serial port 1 ● 2 = Serial port 2 ● 3 = Ethernet / Modbus TCP
i_iId	INT	Device identifier of targeted external device: ● Serial line slave address: 1...247 or ● Ethernet index: 1...16 (Depends on the link configuration).
i_iTimeout	INT	Set the maximum time to wait to receive a response from the modem. Specified in ms. A value of 0 means no timeout is enforced.

Input	Data type	Description
i_diFirstReadObj	DINT	Address of the first object from which values are read.
i_iReadQuantity	INT	Number of objects to read.
i_pbyReadBuffer	POINTER TO BYTE	Address of the buffer in which read object values are stored.
i_diFirstWriteObj	DINT	Address of the first object from which values are written.
i_iWriteQuantity	INT	Number of objects to write.
i_pbyWriteBuffer	POINTER TO BYTE	Address of the buffer in which object values to write are stored.
i_astRemoteDevices	ARRAY [1..Gc_uiRemoteAddressTableLen] OF ST_RemoteDevice	Contains the list of remote device addresses for Modbus TCP communication.

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data type	Description
q_xDone	BOOL	q_xDone is set to TRUE when the function is completed successfully.
q_xBusy	BOOL	q_xBusy is set to TRUE while the function is ongoing.
q_xAborted	BOOL	q_xAborted is set to TRUE when the function is stopped with the i_xAbort input.
q_xError	BOOL	q_xError is set to TRUE when the function stops because of a detected error.
q_etCommError	SEN.CommunicationErrorCodes	q_etCommError contains the value of communication error codes SEN.CommunicationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.
q_etOperError	SEN.OperationErrorCodes	q_etOperError contains the value of operation error codes SEN.OperationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.

FB_SendRecvMsg: Send and Receive Message Function Block

Overview

The following graphic shows the pin diagram of the function block FB_SendRecvMsg:



The function block FB_SendRecvMsg sends and receives user-defined messages. It sends a message on a serial line and then waits for a response. It is also possible to either send without waiting for a response or to receive a message without sending one. This function should be used with an ASCII manager.

I/O Variables Description

The table describes the input variables of the function block in the TwidoEmulationSupport library:

Input	Data type	Description
i_xExecute	BOOL	The function is executed on the rising edge of this input. **NOTE:** When i_xExecute is set to TRUE at the first task cycle in RUNNING after a cold or warm reset, the rising edge is not detected.
i_xAbort	BOOL	Stops the ongoing operation at the rising edge.
i_iLink	INT	Indicates the port used to communicate: ● 1 = Serial port 1 ● 2 = Serial port 2
i_iTimeout	INT	Set the maximum time to wait to receive a response. Specified in ms. A value of 0 means no timeout is enforced.
i_iQuantityToSend	INT	Number of objects to send.
i_pbyBufferToSend	POINTER TO BYTE	Address of the buffer in which object values to send are stored. The first 2 words are control words.
i_iSizeRecvBuffer	INT	Number of objects to receive.
i_pbyBufferToRecv	POINTER TO BYTE	Address of the buffer in which received object values are stored.

The table describes the output variables of the function block in the TwidoEmulationSupport library:

Output	Data type	Description
q_xDone	BOOL	q_xDone is set to TRUE when the function is completed successfully.
q_xBusy	BOOL	q_xBusy is set to TRUE while the function is ongoing.
q_Aborted	BOOL	q_Aborted is set to TRUE when the function is stopped with the i_xAbort input.
q_xError	BOOL	q_xError is set to TRUE when the function stops because of a detected error.
q_etCommError	SEN.CommunicationErrorCodes	q_etCommError contains the value of communication error codes SEN.CommunicationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.
q_etOperError	SEN.OperationErrorCodes	q_etOperError contains the value of operation error codes SEN.OperationErrorCodes *(see EcoStruxure Machine Expert, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.
q_iQuantiyRecv	INT	Number of received objects

Chapter 5
Twido Functions

Twido Functions

Overview

The following list provides an overview of the Twido functions:

- FC_AsciiCharToInt *(see page 46)*
- FC_AsciiToInt *(see page 46)*
- FC_AsciiToReal *(see page 47)*
- FC_ConcatOfInt *(see page 47)*
- FC_CopyArrDint *(see page 48)*
- FC_CopyArrInt *(see page 48)*
- FC_CopyArrReal *(see page 48)*
- FC_CopyBitString *(see page 48)*
- FC_CopyPackedBitString *(see page 49)*
- FC_DegToRad *(see page 49)*
- FC_EqualArrDint *(see page 49)*
- FC_EqualArrReal *(see page 50)*
- FC_FindEqDint *(see page 50)*
- FC_FindEqReal *(see page 50)*
- FC_FindGtDint *(see page 51)*
- FC_FindGtReal *(see page 51)*
- FC_FindLtDint *(see page 51)*
- FC_FindLtReal *(see page 52)*
- FC_HiOfDint *(see page 52)*
- FC_IntToAscii *(see page 52)*
- FC_Lkup *(see page 53)*
- FC_LoOfDint *(see page 54)*
- FC_MaxArrDint *(see page 54)*
- FC_MaxArrReal *(see page 54)*
- FC_MeanArrReal *(see page 55)*
- FC_MinArrDint *(see page 55)*
- FC_MinArrReal *(see page 55)*
- FC_MoveArrDint *(see page 55)*
- FC_MoveArrInt *(see page 56)*
- FC_MoveArrReal *(see page 56)*
- FC_OccurArrDint *(see page 56)*

- FC_OccurArrReal *(see page 56)*
- FC_RadToDeg *(see page 57)*
- FC_RealToAscii *(see page 57)*
- FC_RolArrDint *(see page 58)*
- FC_RolArrReal *(see page 58)*
- FC_RorArrDint *(see page 58)*
- FC_RorArrReal *(see page 59)*
- FC_Round *(see page 59)*
- FC_SortArrDint *(see page 59)*
- FC_SortArrReal *(see page 60)*
- FC_SumArrDint *(see page 60)*
- FC_SumArrReal *(see page 61)*

FC_AsciiCharToInt

This function returns the 2 characters as an integer value.

Input values	i_sChar: STRING with 2 characters.
Return values	Returns characters as integer value.

If the string is invalid, the global variable G_xSystemBitS18 is set to TRUE.

FC_AsciiToInt

The ASCII to integer conversion function executes the conversion of an ASCII string value to its equivalent integer value.

Name in EcoStruxure Machine Expert - Basic / Twido	ASCII_TO_INT
Input values	i_psStartAddr: POINTER TO STRING;
Return values	ASCII_TO_INT: INT

The ASCII to integer instruction rules are as follows:

- The ASCII value must be included between -32768 to 32767.
- The function always reads the most significant byte first.
- Any ASCII character that is not in the interval [0...9] ([30 hex - 39 hex]) is considered to be an end character, except for a minus sign '-' (2D hex) when it is placed as the first character.
- In case of an overflow (>32767 or <-32768), the global variable G_xSystemBitS18 (arithmetic overflow or detected error) is set to 1 and the value 32767 or -32768 is returned.
- If the first character of the operand is an "end" character, the value 0 is returned and the global variable G_xSystemBitS18 is set to TRUE.

FC_AsciiToReal

The ASCII to REAL function executes the conversion of the ASCII string value to its equivalent REAL value.

Name in EcoStruxure Machine Expert - Basic / Twido	ASCII_TO_FLOAT
Input values	i_psStartAddr: POINTER TO STRING
Return values	FC_AsciiToReal : REAL

ASCII to REAL conversion rules are as follows:

- The function reads the most significant byte first.
- Any ASCII character that is not in the interval [0...9] ([30 hex - 39 hex]) is considered to be "end" character, except for:
  - decimal point '.' (2E hex)
  - minus '-' (2D hex)
  - plus '+' (2B hex)
  - exponent 'e' or 'E' (65 hex or 45 hex)
- The ASCII string format can be in exponential notation (that is "-2.34567e+13") or in decimal notation (that is 9826.3457).
- In case of an overflow (calculation result is > 3.402824E+38 or < -3.402824E+38):
  - The global variable G_xSystemBitS18 (arithmetic overflow or detected error) is set to TRUE.
  - The value +/- 1.#INF (+ or - infinite value) is returned.
- If the calculation result is between -1.175494E-38 and 1.175494E-38, then the result is rounded to 0.0.
- If the operand is not a number, the value 1.#QNAN is returned and the global variable G_xSystemBitS18 is set to TRUE.

FC_ConcatOfInt

This function concatenates two integers into a double integer.

Name in EcoStruxure Machine Expert - Basic / Twido	CONCATW
Input values	i_iLowVal: INT i_iHighVal: INT
Return values	FC_ConcatOfInt : DINT

FC_CopyArrDint

The function copies an array of DINT values to another given memory address.

Name in EcoStruxure Machine Expert - Basic / Twido	COPY_ARR_DINT
Input values	i_pdiSource : POINTER TO DINT i_pdiDestination : POINTER TO DINT i_bySize : BYTE
Return values	FC_CopyArrDint : BOOL

FC_CopyArrInt

The function copies an array of INT values to another given memory address.

Name in EcoStruxure Machine Expert - Basic / Twido	COPY_ARR_INT
Input values	i_piSource : POINTER TO INT i_piDestination: POINTER TO INT i_bySize : BYTE
Return values	FC_CopyArrInt : BOOL

FC_CopyArrReal

The function copies an array of REAL values to another given memory address.

Name in EcoStruxure Machine Expert - Basic / Twido	COPY_ARR_REAL
Input values	i_prSource : POINTER TO REAL i_prDestination : POINTER TO REAL i_bySize : BYTE
Return values	FC_CopyArrReal : BOOL

FC_CopyBitString

The function copies an array of BOOL values to another given memory address.

Name in EcoStruxure Machine Expert - Basic / Twido	COPY_ARR_BOOL
Input values	i_pxSource : POINTER TO BOOL i_pxDestination : POINTER TO BOOL i_bySize : BYTE
Return values	FC_CopyBitString : BOOL

FC_CopyPackedBitString

The function copies a packed BitString of the length of i_bySize bits to another memory location.

Name in EcoStruxure Machine Expert - Basic / Twido	FC_CopyPackedBitString
Input values	i_pbySource : POINTER TO BYTE i_pbyDestination: POINTER TO BYTE i_bySize : BYTE - limited to SIZE OF (DINT)
Return values	FC_CopyPackedBitString : DINT

FC_DegToRad

The function FC_DegToRad executes the conversion of an angle expressed in degree to radian.

Name in EcoStruxure Machine Expert - Basic / Twido	DEG_TO_RAD
Input values	i_rDeg : REAL;
Return values	FC_DegToRad: REAL
Formula	Radian = Degrees * (Pi / 180) Where, Pi = 3.1415926535

Rules of use:

The angle to be converted must be between -737280.0 and +737280.0.

For values outside these ranges, the displayed result is + 1.#QNAN and the global variable G_xSystemBitS18 is set to TRUE.

FC_EqualArrDint

The function FC_EqualArrDint compares 2 defined size tables, element by element. If there is a difference, the rank of the first dissimilar element is returned in the form of an integer. If there is no difference, the value"-1" is returned.

Name in EcoStruxure Machine Expert - Basic / Twido	EQUAL_ARR
Input values	i_pdiFirstArrStartValue: POINTER TO DINT i_pdiSecondArrStartValue: POINTER TO DINT i_bySize: BYTE
Return values	FC_EqualArrDint: INT

The start addresses of the 2 arrays to be compared are provided at the 2 pointer inputs.

The number of elements to be compared must be defined at the input i_bySize.

FC_EqualArrReal

The function FC_EqualArrReal compares 2 defined size tables, element by element. If there is a difference shown, the rank of the first dissimilar element is returned in the form of an integer. If there is no difference, then the value -1 is returned.

Name in EcoStruxure Machine Expert - Basic / Twido	EQUAL_ARR
Input values	i_pdiFirstArrStartValue: POINTER TO REAL i_pdiSecondArrStartValue: POINTER TO REAL i_bySize: BYTE
Return values	FC_EqualArrReal : INT

The start addresses of the 2 arrays to be compared are provided at the 2 pointer inputs.

The number of elements to be compared must be defined at the input i_bySize.

FC_FindEqDint

The function FC_FindEqDint searches in a set of given DINT values for the position of the first array element, which is equal to a given value.

Name in EcoStruxure Machine Expert - Basic / Twido	FIND_EQR
Input values	i_diValue: DINT i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_FindEqDint : INT

If the given value is not contained in the array, -1 is returned.

FC_FindEqReal

The function FC_FindEqReal searches in a set of given REAL values for the position of the first array element which is equal to a given value defined by the user.

Name in EcoStruxure Machine Expert - Basic / Twido	FIND_EQR
Input values	i_rValue: REAL i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_FindEqReal : INT

If the given value is not contained in the array, -1 is returned.

FC_FindGtDint

The function FC_FindGtDint search in a set of given DINT values for the position of the first array element which is greater than a given value defined by the user.

Name in EcoStruxure Machine Expert - Basic / Twido	FIND_GTR
Input values	i_diValue: DINT i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_FindGtDint : INT

If no element in the array is greater than the given value, -1 is returned.

FC_FindGtReal

The function FC_FindGtReal searches in a set of given REAL values for the position of the first array element which is greater than a given value defined by the user.

Name in EcoStruxure Machine Expert - Basic / Twido	FIND_GTR
Input values	i_rValue: REAL i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_FindGtReal : INT

If no element in the array is greater than the given value, -1 is returned.

FC_FindLtDint

The function FC_FindLtDint searches in a set of given DINT values for the position of the first array element which is smaller than a given value defined by the user.

Name in EcoStruxure Machine Expert - Basic / Twido	FIND_LTR
Input values	i_diValue: DINT i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_FindLtDint : INT

If no element in the array is greater than the given value, -1 is returned.

FC_FindLtReal

The function FC_FindLtReal searches in a set of given REAL values for the position of the first array element which is smaller than a given value defined by the user.

Name in EcoStruxure Machine Expert - Basic / Twido	FIND_LTR
Input values	i_rValue: REAL i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_FindLtReal : INT

If no element in the array is greater than the given value, -1 is returned.

FC_HiOfDint

This function extracts most significant bits (MSB) of a double integer to an integer.

Name in EcoStruxure Machine Expert - Basic / Twido	HW
Input values	i_diVal: DINT
Return values	FC_LoOfDint : INT

FC_IntToAscii

The integer to ASCII function executes the conversion of an integer value to its equivalent ASCII string value.

Name in EcoStruxure Machine Expert - Basic / Twido	INT_TO_ASCII
Input values	i_iVal: INT
Return values	INT_TO_ASCII : ARRAY [1..4] OF INT

The integer to ASCII conversion rules are as follows:

- The integer value must be included between -32768 to 32767.
- The function always writes the most significant byte first.
- The end character is "Carriage return" (ASCII 13).
- The function automatically determines how many %MWs should be filled with ASCII values (from 1 to 4)

FC_Lkup

The function FC_Lkup is used to interpolate a set of X versus Y floating point data for a given X value.

Name in EcoStruxure Machine Expert - Basic / Twido	LKUP
Input values	i_prStartAddr : POINTER TO REAL i_bySize : BYTE
Return values	FC_Lkup : INT

The following conditions apply for the input value i_prSartAddr:

- even number of values
- minimum of 6 values
- first element is x value to be found
- second element is set by the function: interpolation result
- all following elements are interpolation supporting points by pairs of X and Y

Interpolation rules:

The LKUP function uses the linear interpolation rule, as defined in the following equation:

$$Y = Y_i + \left[\frac{(Y_{i+1} - Y_i)}{(X_{i+1} - X_i)} \cdot (X - X_i)\right]$$

For $X_i \leq X \leq X_i + 1$, where i = 1 … (m-1)

Assuming X_i values are ranked in ascending order: $X_1 \leq X_2 \leq ...X...\leq X_{m-1} \leq X_m$

If any of 2 consecutive X_i values are equal ($X_i=X_{i+1}=X$), the equation 1 results in an invalid exception. To handle this exception, the following algorithm is used in place of equation 1:

$$Y = \left[\frac{(Y_{i+1} - Y_i)}{2}\right]$$

For $X_i = X_i+1 = X$, where i = 1…(m-1).

Result value:

The result value shows if the interpolation was successful or not.

0: Successful interpolation

1: Interpolation error: incorrect array, Xm < Xm-1

2: Interpolation error: i_rXValue out of range, X < X1

4: Interpolation error: i_rXValue out of range, X > Xm

8: Invalid size of data array: i_prYValue is set as an odd number, or i_prYValue < 6

The result value does not contain the computed interpolation value (Y). For a given (X) value, the result of the interpolation (Y) is contained in `i_prYValue`.

`i_rXValue` is the floating point variable that contains the user-defined (X) value for which to compute the interpolated (Y) value.

The valid range for `i_rXValue` is:

$X_1 \leq$ `i_rXValue` $\leq X_m$

FC_LoOfDint

This function extracts least significant bits (LSB) of a double integer to an integer.

Name in EcoStruxure Machine Expert - Basic / Twido	`LW`
Input values	`i_diVal`: DINT
Return values	`FC_LoOfDint` : INT

FC_MaxArrDint

In a set of given DINT values, this function searches the maximum value. The search is carried out only on the defined length of the table.

Name in EcoStruxure Machine Expert - Basic / Twido	`MAX_ARR`
Input values	`i_pdiStartAddr`: POINTER TO DINT `i_bySize` : BYTE
Return values	`FC_MaxArrDint` : DINT

FC_MaxArrReal

In a set of given REAL values, this function searches the maximum value. The search is only carried out within the defined table length.

Name in EcoStruxure Machine Expert - Basic / Twido	`MAX_ARR`
Input values	`i_prStartAddr`: POINTER TO REAL `i_bySize` : BYTE
Return values	`FC_MaxArrReal` : REAL

FC_MeanArrReal

In a set of given REAL values, this function is used to calculate the mean average in the required length of REAL values table.

Name in EcoStruxure Machine Expert - Basic / Twido	MEAN
Input values	i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_MeanArrReal : REAL

FC_MinArrDint

In a set of given DINT values, this function searches the minimum value. The search is carried out only on the defined length of the table.

Name in EcoStruxure Machine Expert - Basic / Twido	MIN_ARR
Input values	i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_MinArrDint : DINT

FC_MinArrReal

In a set of given REAL values, this function searches the minimum value. The search is carried out only on the defined length of the table.

Name in EcoStruxure Machine Expert - Basic / Twido	MAX_ARR
Input values	i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_MinArrReal : REAL

FC_MoveArrDint

The function moves the input value in the DINT value tables for a number of elements equal to a given value.

Name in EcoStruxure Machine Expert - Basic / Twido	MOVE_ARR_DINT
Input values	i_diValue : DINT i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_MoveArrDint : BOOL

FC_MoveArrInt

The function moves the input value in the INT values table for a number of elements equal to a given value.

Name in EcoStruxure Machine Expert - Basic / Twido	MOVE_ARR_INT
Input values	i_iValue : INT i_piStartAddr: POINTER TO INT i_bySize : BYTE
Return values	FC_MoveArrInt : BOOL

FC_MoveArrReal

The function moves the input value in the REAL value tables for a number of elements equal to a given value.

Name in EcoStruxure Machine Expert - Basic / Twido	MOVE_ARR_REAL
Input values	i_rValue : REAL i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_MoveArrReal : BOOL

FC_OccurArrDint

In a set of given DINT values, this function searches the number of elements equal to a given value.

Name in EcoStruxure Machine Expert - Basic / Twido	OCCUR_ARR
Input values	i_diValue: DINT i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_OccurArrDint : INT

FC_OccurArrReal

In a set of given REAL values, this function searches the number of elements equal to a given value.

Name in EcoStruxure Machine Expert - Basic / Twido	OCCUR_ARR

Input values	i_rValue: REAL i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_OccurArrReal : INT

FC_RadToDeg

This function FC_RadToDeg executes the conversion of an angle expressed in radian to degree.

Name in EcoStruxure Machine Expert - Basic / Twido	RAD_TO_DEG
Input values	i_rRad : REAL;
Return values	FC_RadToDeg: REAL
Formula	Degree = Radian * (180 / Pi) Where, Pi = 3.1415926535

Rules of use:

The angle to be converted must be between -4096Pi and 4096Pi.

For values outside these ranges, the displayed result is + 1.#QNAN and G_xSystemBitS18 is set to TRUE.

FC_RealToAscii

The REAL to ASCII function executes the conversion of REAL value to its equivalent ASCII string value.

Name in EcoStruxure Machine Expert - Basic / Twido	FLOAT_TO_ASCII
Input values	i_rVal: REAL
Return values	FC_RealToAscii : ARRAY [1..7] OF INT

The REAL to ASCII conversion rules are as follows:

- The function always writes the most significant byte (MSB) first.
- The representation is made using conventional exponential notation.
- "Infinite" or "Not a number" results return the string "NAN".
- The end character is "Carriage return" (ASCII 13).
- The conversion precision is 6 figures.

FC_RolArrDint

In a given DINT array this function shifts every element n position(s) towards the start address of the list. The first n elements are moved to the end of the list.

Name in EcoStruxure Machine Expert - Basic / Twido	ROL_ARR
Input values	i_iShiftPosNumber: INT i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_RolArrDint : BOOL

If the value of i_iShiftPosNumber is negative or 0, no shift is performed.

FC_RolArrReal

In a set of given REAL values, this function performs a rotate shift of n position(s) from top to bottom of the elements.

Name in EcoStruxure Machine Expert - Basic / Twido	ROL_ARR
Input values	i_iShiftPosNumber: INT i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_RolArrReal : BOOL

If the value of i_iShiftPosNumber is negative or 0, no shift is performed.

FC_RorArrDint

In a given DINT array this function shifts every element n position(s) towards the list's end address. The last n elements are moved to the begin of the list.

Name in EcoStruxure Machine Expert - Basic / Twido	ROR_ARR
Input values	i_iShiftPosNumber: INT i_pdiStartAddr: POINTER TO DINT i_bySize : BYTE
Return values	FC_RorArrDint : BOOL

If the value of i_iShiftPosNumber is negative or 0, no shift is performed.

FC_RorArrReal

In a set of given REAL values, this function performs a rotate shift of n position(s) from bottom to top of the elements.

Name in EcoStruxure Machine Expert - Basic / Twido	ROL_ARR
Input values	i_iShiftPosNumber: INT i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_RorArrReal : BOOL

If the value of i_iShiftPosNumber is negative or 0, no shift is performed.

FC_Round

The function FC_Round rounds a floating point representation that is stored in an ASCII string.

Name in EcoStruxure Machine Expert - Basic / Twido	ROUND
Input values	i_psStartAddr: POINTER TO STRING; i_byRoundNumber: BYTE;
Return values	FC_Round: ARRAY[1..7] of INT

The ROUND instruction rules are as follows:

- The operand is always rounded down.
- The end character of the operand string is used as an end character for the result string.
- The end character can be any ASCII character that is in the interval [0...9] ([30 hex - 39 hex]), except:
  - dot '.' (2E hex),
  - minus '-' (2D hex),
  - plus '+' (2B hex),
  - Exp 'e' or 'E' (65 hex or 45 hex).
- The result and operand must not be longer than 13 bytes: Maximum size of an ASCII string is 13 bytes.

FC_SortArrDint

The function sorts the elements of a DINT values table in ascending or descending order and stores the result in the same table.

Name in EcoStruxure Machine Expert - Basic / Twido	SORT_ARR

Input values	i_iSortDirection: INT i_pdiStartAddr: POINTER TO DINT i_bySize: BYTE
Return values	FC_SortArrDint: BOOL

The direction parameter i_iSortDirection indicates the sort sequence:
- direction > 0, sorting in ascending order.
- direction < 0, sorting in descending order.
- direction = 0, no sorting is performed.

FC_SortArrReal

The function sorts the elements of a DINT values table in ascending or descending order and stores the result in the same table.

Name in EcoStruxure Machine Expert - Basic / Twido	SORT_ARR
Input values	i_iSortDirection: INT i_prStartAddr: POINTER TO REAL i_bySize : BYTE
Return values	FC_SortArrReal : BOOL

The direction parameter i_iSortDirection gives the order of the sort:
- direction > 0, sorting in ascending order.
- direction < 0, sorting in descending order.
- direction = 0, no sorting is performed.

FC_SumArrDint

The function FC_SumArrDint is used to perform the addition of a defined number of array elements whose address is defined at the input i_pdiStartAddr. This means that the function sums up all the elements of an object table.

Name in EcoStruxure Machine Expert - Basic / Twido	SUM_ARR
Input values	i_pdiStartAddr: POINTER TO DINT i_bySize: BYTE
Return values	FC_SumArrDint : DINT

When the result is not within the valid double word format range according to the table operand, the global variable G_xSystemBitS18 is set to TRUE.

FC_SumArrReal

The function FC_SumArrReal is used to perform the addition of a defined number of array elements whose address is defined at the input i_pdiStartAddr. This means that the function sums up all the elements of an object table.

Name in EcoStruxure Machine Expert - Basic / Twido	SUM_ARR
Input values	i_pdiStartAddr: POINTER TO DINT i_bySize: BYTE
Return values	FC_SumArrReal : REAL

When the result is not within the valid double word format range according to the table operand, the global variable G_xSystemBitS18 is set to TRUE.

Glossary

!

%MW

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

F

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

A programming unit that has 1 or more inputs and returns 1 or more outputs. FBs are called through an instance (function block copy with dedicated name and variables) and each instance has a persistent state (outputs and internal variables) from 1 call to the other.

Examples: timers, counters

L

LSB

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

M

MSB

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

P

PID

(*proportional, integral, derivative*) A generic control loop feedback mechanism (controller) widely used in industrial control systems.

PWM

(*pulse width modulation*) A fast output that oscillates between off and on in an adjustable duty cycle, producing a rectangular wave form (though you can adjust it to produce a square wave).

Index

A

ASCII_TO_FLOAT, *47*
ASCII_TO_INT, *46*

C

CONCATW, *47*
COPY_ARR_BOOL, *48*
COPY_ARR_DINT, *48*
COPY_ARR_INT, *48*
COPY_ARR_REAL, *48*

D

DEG_TO_RAD, *49*

E

EQUAL_ARR, *49*, *50*

F

FB_Counter, *18*
FB_Drum, *25*
FB_EXCH, *34*
FB_FiFo, *19*
FB_LiFo, *19*
FB_PID, *29*
FB_ReadVar, *37*
FB_ScheduleBlock, *22*
FB_SendRecvMsg, *43*
FB_ShiftBitRegister, *20*
FB_StepCounter, *28*
FB_Timer, *16*
FB_WriteReadVar, *41*
FB_WriteVar, *39*
FC_AsciiCharToInt, *46*
FC_AsciiToInt, *46*
FC_AsciiToReal, *47*
FC_ConcatOfInt, *47*
FC_CopyArrDint, *48*
FC_CopyArrInt, *48*
FC_CopyArrReal, *48*
FC_CopyBitString, *48*
FC_CopyPackedBitString, *49*, *49*
FC_DegToRad, *49*
FC_EqualArrDint, *49*
FC_EqualArrReal, *50*
FC_FindEqDint, *50*
FC_FindEqReal, *50*
FC_FindGtDint, *51*
FC_FindGtReal, *51*
FC_FindLtDint, *51*
FC_FindLtReal, *52*
FC_HiOfDint, *52*
FC_IntToAscii, *52*
FC_Lkup, *53*
FC_LoOfDint, *54*
FC_MaxArrDint, *54*
FC_MaxArrReal, *54*
FC_MeanArrReal, *55*
FC_MinArrDint, *55*
FC_MinArrReal, *55*
FC_MoveArrDint, *55*
FC_MoveArrInt, *56*
FC_MoveArrReal, *56*
FC_OccurArrDint, *56*
FC_OccurArrReal, *56*
FC_RadToDeg, *57*
FC_RealToAscii, *57*
FC_RolArrDint, *58*
FC_RolArrReal, *58*
FC_RorArrDint, *58*
FC_RorArrReal, *59*
FC_Round, *59*
FC_SortArrDint, *59*
FC_SortArrReal, *60*
FC_SumArrDint, *60*
FC_SumArrReal, *61*
FIND_EQR, *50*, *50*
FIND_GTR, *51*, *51*

FIND_LTR, *51*, *52*
FLOAT_TO_ASCII, *57*

H

HW, *52*

I

INT_TO_ASCII, *52*

L

LKUP, *53*
LW, *54*

M

MAX_ARR, *54*, *54*, *55*
MEAN, *55*
MIN_ARR, *55*
MOVE_ARR_DINT, *55*
MOVE_ARR_INT, *56*
MOVE_ARR_REAL, *56*

O

OCCUR_ARR, *56*, *56*

R

RAD_TO_DEG, *57*
ROL_ARR, *58*, *58*, *59*
ROR_ARR, *58*
ROUND, *59*

S

SORT_ARR, *59*, *60*
SUM_ARR, *60*, *61*

T

TwidoEmulationSupport library
- FB_Counter, *18*
- FB_Drum, *25*
- FB_EXCH, *34*
- FB_FiFo, *19*
- FB_LiFo, *19*
- FB_PID, *29*
- FB_ReadVar, *37*
- FB_ScheduleBlock, *22*
- FB_SendRecvMsg, *43*
- FB_ShiftBitRegister, *20*
- FB_StepCounter, *28*
- FB_Timer, *16*
- FB_WriteReadVar, *41*
- FB_WriteVar, *39*