

Modicon M238 Logic Controller

System Functions and Variables M238 PLC System Library Guide

11/2015

E100000000364.09

www.schneider-electric.com

Schneider
 Electric

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

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

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

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

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

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

© 2015 Schneider Electric. All rights reserved.

Table of Contents



	Safety Information	5
	About the Book.	7
Chapter 1	M238 System Variables	11
1.1	System Variables: Definition and Use	12
	Understanding System Variables	13
	Using System Variables	15
1.2	PLC_R and PLC_W Structures	17
	PLC_R: Controller Read-Only System Variables	18
	PLC_W: Controller Read/Write System Variables	20
1.3	SERIAL_R and SERIAL_W Structures	21
	SERIAL_R[0...1]: Serial Line Read-Only System Variables	22
	SERIAL_W[0...1]: Serial Line Read/Write System Variables	23
Chapter 2	M238 System Functions.	25
2.1	M238 Read Functions	26
	GetBatteryLevel: Returns Remaining Power Charge of the Battery	27
	GetEventsNumber: Returns the Number of External Events Detected	28
	GetFirmwareVersion: Returns Information About the Firmware, the Boot and the Coprocessor Versions	29
	GetHardwareVersion: Returns the Hardware Version	30
	GetLastStopCause: Returns the Cause Last Stop	31
	GetLastStopTime: Returns the Date and Time of the Last Detected Stop	32
	GetLocalIOStatus: Returns the Embedded I/O Status	33
	GetPlcFault: Returns Detected Errors on the Controller I/O	35
	GetRightBusStatus: Returns the Status of the Expansion Bus	36
	GetSerialNumber: Returns the Serial Number of the Controller	40
	GetShortCutStatus: Returns the Short-Circuit Status on Embedded Outputs	41
	IsFirstMastColdCycle: Indicates if Cycle is the First MAST Cold Start Cycle	42
	IsFirstMastCycle: Indicates if Cycle is the First MAST Cycle	43
	IsFirstMastWarmCycle: Indicates if Cycle is the First MAST Warm Start Cycle	45

2.2	M238 Write Functions	46
	InhibitBatLowLed: Disables or Re-enables the Batt LED.....	47
	ResetEventsNumber: Resets Events Number.....	48
	SetRTCDrift: Adjust the Real Time Clock Each Week.....	49
Chapter 3	M238 PLCSystem Library Data Types	51
3.1	PLC_R/W System Variables Data Types.....	52
	PLC_R_STATUS: Controller Status Codes.....	53
	PLC_R_APPLICATION_ERROR: Detected Application Error Status Codes.....	54
	PLC_R_BOOT_PROJECT_STATUS: Boot Project Status Codes ...	56
	PLC_R_IO_STATUS: I/O Status Codes	57
	PLC_R_STOP_CAUSE: From RUN State to Other State Transition Cause Codes.....	58
	PLC_W_COMMAND: Control Command Codes.....	59
3.2	System Function Data Types.....	60
	FIRMWARE_VERSION: GetFirmwareVersion Function Output Type.....	61
	BOOT_PROJECT_STATUS: GetBootProjectStatus Function Output Codes.....	62
	STOP_WHY: GetLastStopCause Function Output Codes	63
	LOCAL_IO_GET_STATUS: GetLocalIOStatus Function Parameter Codes.....	64
	LOCAL_IO_GEN_STATUS: GetLocalIOStatus Function Output Codes.....	65
	RIGHTBUS_GET_STATUS: GetRightBusStatus Function Parameter Codes.....	66
	DAY_OF_WEEK: SetRTCDrift Function Day Parameter Codes...	67
	HOUR: SetRTCDrift Function Hour Parameter Type	68
	MINUTE: SetRTCDrift Function Minute Parameter Type	69
Appendices	71
Appendix A	Function and Function Block Representation	73
	Differences Between a Function and a Function Block	74
	How to Use a Function or a Function Block in IL Language	75
	How to Use a Function or a Function Block in ST Language.....	79
Glossary	83
Index	89

Safety Information



Important Information

NOTICE

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



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



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

DANGER

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

WARNING

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

CAUTION

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

NOTICE

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

PLEASE NOTE

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

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

About the Book



At a Glance

Document Scope

This document will acquaint you with the system functions and variables offered within the Modicon M238 Logic Controller. The M238 PLCSystem library contains functions and variables to get information and send commands to the controller system.

This document describes the data types functions and variables of the M238 PLCSystem library.

The following basic knowledge is required:

- basic information on functionality, structure and configuration of the M238
- programming in the FBD, LD, ST, IL or CFC language
- System Variables (global variables)

Validity Note

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

Related Documents

Title of Documentation	Reference Number
M238 Logic Controller Programming Guide	EIO0000000384 (ENG); EIO0000000385 (FRE); EIO0000000386 (GER); EIO0000000387 (ITA); EIO0000000388 (SPA); EIO0000000389 (CHS)

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

Product Related Information

WARNING

LOSS OF CONTROL

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

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

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

WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2004/108/EC	Electromagnetic Compatibility Directive
2006/95/EC	Low Voltage Directive

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

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

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

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

Chapter 1

M238 System Variables

Overview

This chapter:

- gives an introduction to the system variables (*see page 12*)
- describes the system variables (*see page 18*) included with the M238 PLCSystem library

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
1.1	System Variables: Definition and Use	12
1.2	PLC_R and PLC_W Structures	17
1.3	SERIAL_R and SERIAL_W Structures	21

Section 1.1

System Variables: Definition and Use

Overview

This section defines system variables and how to implement them in the Modicon M238 Logic Controller.

What Is in This Section?

This section contains the following topics:

Topic	Page
Understanding System Variables	13
Using System Variables	15

Understanding System Variables

Introduction

This section describes how system variables are implemented. System variables:

- allow you to access general system information, perform system diagnostics, and command simple actions.
- are structured variables conforming to IEC 61131-3 definitions and naming conventions. You can access the system variables using the IEC symbolic name `PLC_GVL`. Some of the `PLC_GVL` variables are read-only (for example, `PLC_R`) and some are read/write (for example, `PLC_W`).
- are automatically declared as global variables. They have system-wide scope and can be accessed by any Program Organization Unit (POU) in any task.

Naming Convention

The system variables are identified by:

- a structure name that represents the category of system variable. For example, `PLC_R` represents a structure name of read-only variables used for the controller diagnostic.
- a set of component names that identifies the purpose of the variable. For example, `i_wVendorID` represents the controller vendor ID.

You can access the system variables by typing the structure name of the variables followed by the name of the component.

Here is an example of system variable implementation:

```
VAR
    myCtr_Serial : DWORD;
    myCtr_ID : DWORD;
    myCtr_FramesRx : UDINT;
END_VAR

myCtr_Serial := PLC_R.i_dwSerialNumber;
myCtr_ID := PLC_R.i_wVendorID;
myCtr_FramesRx := SERIAL_R[0].i_udiFramesReceivedOK;
```

NOTE: The fully qualified name of the system variable in the example above is `PLC_GVL.PLC_R.i_wVendorID`. The `PLC_GVL` is implicit when declaring a variable using the **Input Assistant**, but it may also be entered in full. Good programming practice often dictates the use of the fully qualified variable name in declarations.

System Variables Location

2 kinds of system variables are defined for use when programming the controller:

- located variables
- unlocated variables

The located variables:

- have a fixed location in a static %MW area: %MW60000 to %MW60199 for read-only system variables.
- are accessible through Modbus serial request both in RUNNING and STOPPED states.
- are used in SoMachine programs according to the `structure_name.component_name` convention explained previously. %MW addresses from 0 to 59999 can be accessed directly. Addresses greater than this are considered out of range by SoMachine and can only be accessed through the `structure_name.component_name` convention.

The unlocated variables:

- are not physically located in the %MW area.
- are not accessible through any fieldbus or network requests unless you locate them in the relocation table, and only then these variables can be accessed in RUNNING and STOPPED states. The relocation table uses the following dynamic %MW areas:
 - %MW60200 to %MW61999 for read-only variables
 - %MW62200 to %MW63999 for read/write variables
- are used in SoMachine programs according to the `structure_name.component_name` convention explained previously. %MW addresses from 0 to 59999 can be accessed directly. Addresses greater than this are considered out of range by SoMachine and can only be accessed through the `structure_name.component_name` convention.

Using System Variables

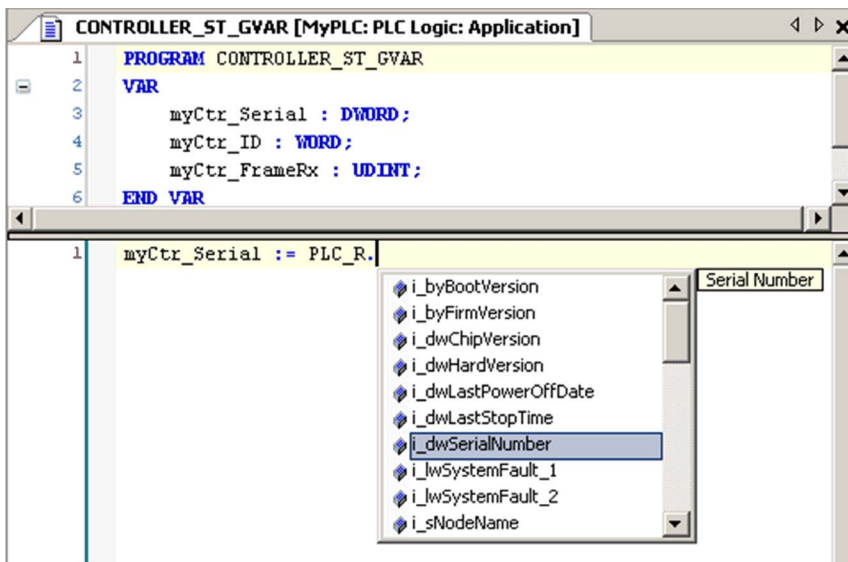
Introduction

This section describes the steps required to program and to use system variables in SoMachine. System variables are global in scope, and you can use them in all the Program Organization Units (POUs) of the application.

System variables do not need to be declared in the Global Variable List (GVL). They are automatically declared from the controller system library.

Using System Variables in a POU

SoMachine has an auto-completion feature. In a **POU**, start by entering the system variable structure name (PLC_R, PLC_W...) followed by a dot. The system variables appear in the **Input Assistant**. You can select the desired variable or enter the full name manually.



NOTE: In the example above, after you type the structure name `PLC_R.`, SoMachine offers a pop-up menu of possible component names/variables.

Example

The following example shows the use of some system variables:

```
VAR
    myCtr_Serial : DWORD;
    myCtr_ID : WORD;
    myCtr_FramesRx : UDINT;
END_VAR

myCtr_Serial := PLC_R.i_dwSerialNumber;
myCtr_ID := PLC_R.i_wVendorID;
myCtr_FramesRx := SERIAL_R[0].i_udiFramesReceivedOK;
```

Section 1.2

PLC_R and PLC_W Structures

Overview

This section lists and describes the different system variables included in the `PLC_R` and `PLC_W` structures.

What Is in This Section?

This section contains the following topics:

Topic	Page
<code>PLC_R</code> : Controller Read-Only System Variables	18
<code>PLC_W</code> : Controller Read/Write System Variables	20

PLC_R: Controller Read-Only System Variables

Variable Structure

This table describes the parameters of the PLC_R system variable (PLC_R_STRUCT type):

Modbus Address ⁽¹⁾	Var Name	Type	Comment
60000	i_wVendorID	WORD	Controller Vendor ID. 101A hex = Schneider Electric
60001	i_wProductID	WORD	Controller Reference ID. NOTE: Vendor ID and Reference ID are the components of the Target ID of the controller displayed in the communication settings view (Target ID = 101A XXXX hex).
60002	i_dwSerialNumber	DWORD	Controller Serial Number
60004	i_byFirmVersion[0..3]	ARRAY[0..3] OF BYTE	Controller Firmware Version [aa.bb.cc.dd]: <ul style="list-style-type: none"> ● i_byFirmVersion[0] = aa ● ... ● i_byFirmVersion[3] = dd
60006	i_byBootVersion[0..3]	ARRAY[0..3] OF BYTE	Controller Boot Version [aa.bb.cc.dd]: <ul style="list-style-type: none"> ● i_byBootVersion[0] = aa ● ... ● i_byBootVersion[3] = dd
60008	i_dwHardVersion	DWORD	Controller Hardware Version.
60010	i_dwChipVersion	DWORD	Controller Coprocessor Version.
60012	i_wStatus	PLC_R_STATUS (see page 53)	State of the controller.
60013	i_wBootProjectStatus	PLC_R_BOOT_PROJECT_STATUS (see page 56)	Returns information about the boot application stored in FLASH memory.
60014	i_wLastStopCause	PLC_R_STOP_CAUSE (see page 58)	Cause of the last transition from RUN to another state.
60015	i_wLastApplicationError	PLC_R_APPLICATION_ERROR (see page 54)	Cause of the last controller exception.
60016	i_lwSystemFault_1	LWORD	Bit field FFFF FFFF FFFF FFFF hex indicates no error detected. If TM2 error is detected, the bit 1 is FALSE.
60020	i_lwSystemFault_2	LWORD	Not used.
60024	i_wIOStatus1	PLC_R_IO_STATUS (see page 57)	Embedded I/O status.

Modbus Address ⁽¹⁾	Var Name	Type	Comment
60025	i_wIOStatus2	PLC_R_IO_STATUS (see page 57)	TM2 I/O status.
60026	i_wBatteryStatus	WORD	Charge remaining in the battery. This system variable can take the following significant values: <ul style="list-style-type: none"> ● FFFF hex = 100% = 3.5 V ● 7FFF hex = 50% = 2.75 V ● 0000 hex = 0% = 2 V Flashing thresholds of the Batt LED: <ul style="list-style-type: none"> ● 100% = always OFF ● 50% = periodic flashing ● 0% = always ON
60028	i_dwAppliSignature1	DWORD	First DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
60030	i_dwAppliSignature2	DWORD	Second DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
60032	i_dwAppliSignature3	DWORD	Third DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
60034	i_dwAppliSignature4	DWORD	Fourth DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
⁽¹⁾ Not accessible through the application.			

PLC_W: Controller Read/Write System Variables

Variable Structure

This table describes the parameters of the PLC_W system variable (PLC_W_STRUCT type):

%MW	Var Name	Type	Comment
n/a	q_uiOpenPLCControl	UINT	When Value passes from 0 to 6699, the command previously written in the following PLC_W.q_wPLCControl is executed.
n/a	q_wPLCControl	PLC_W_COMMAND (see page 59)	Controller RUN / STOP command executed when the system variable PLC_R.q_uiOpenPLCControl value passes from 0 to 6699.

NOTE: n/a means that there is no pre-defined %MW mapping for this system variable.

Section 1.3

SERIAL_R and SERIAL_W Structures

Overview

This section lists and describes the different system variables included in the SERIAL_R and SERIAL_W structures.

What Is in This Section?

This section contains the following topics:

Topic	Page
SERIAL_R[0...1]: Serial Line Read-Only System Variables	22
SERIAL_W[0...1]: Serial Line Read/Write System Variables	23

SERIAL_R[0...1]: Serial Line Read-Only System Variables

Introduction

SERIAL_R is an array of 2 SERIAL_R_STRUCT type. Each element of the array returns diagnostic system variables for the corresponding serial line.

For the M238 Logic Controller:

- Serial_R[0] refers to the serial line 1
- Serial_R[1] refers to the serial line 2 (only for TM238LFDC24DT and TM238LFAC24DR)

Variable Structure

This table describes the parameters of the SERIAL_R[0...1] system variables:

%MW	Var Name	Type	Comment
Serial Line			
n/a	i_udiFramesTransmittedOK	UDINT	Number of frames successfully transmitted.
n/a	i_udiFramesReceivedOK	UDINT	Number of frames received without any errors detected.
n/a	i_udiRX_MessagesError	UDINT	Number of frames received with errors detected (checksum, parity).
Modbus Specific			
n/a	i_uiSlaveExceptionCount	UINT	Number of Modbus exception responses returned by the logic controller.
n/a	i_udiSlaveMsgCount	UINT	Number of messages received from the Master and addressed to the logic controller.
n/a	i_uiSlaveNoRespCount	UINT	Number of Modbus broadcast requests received by the logic controller.
n/a	i_uiSlaveNakCount	UINT	Not used
n/a	i_uiSlaveBusyCount	UINT	Not used
n/a	i_uiCharOverrunCount	UINT	Number of character overruns.
n/a means that there is no predefined %MW mapping for this system variable. Not used means that the variable is not maintained by the system, and that if the value of the variable is non-zero, it should be considered extraneous.			

The SERIAL_R counters are reset on:

- Download.
- Controller reset.
- SERIAL_W[x].q_wResetCounter command.
- Reset command by Modbus request function code number 8.

SERIAL_W[0...1]: Serial Line Read/Write System Variables

Introduction

SERIAL_W is an array of 2 SERIAL_W_STRUCT type. Each element of the array resets the SERIAL_R system variables for the corresponding serial line to be reset.

For the M238 Logic Controller:

- Serial_W[0] refers to the serial line 1
- Serial_W[1] refers to the serial line 2 (only for TM238LFDC24DT and TM238LFAC24DR)

Variable Structure

This table describes the parameters of the SERIAL_W[0...1] system variable:

%MW	Var Name	Type	Comment
n/a	q_wResetCounter	WORD	Transition from 0 to 1 resets all SERIAL_R[0...1] counters. To reset the counters again, it is necessary to write this register to 0 before another transition from 0 to 1 can take place.

NOTE: n/a means that there is no predefined %MW mapping for this system variable.

Chapter 2

M238 System Functions

Overview

This chapter describes the system functions included in the M238 PLCSystem library.

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
2.1	M238 Read Functions	26
2.2	M238 Write Functions	46

Section 2.1

M238 Read Functions

Overview

This section describes the read functions included in the M238 PLCSystem library.

In many cases, similar information returned by these functions can also be obtained through System Variables (*see page 13*). System Variables are common across the control platforms, and therefore it is recommended to use them where possible for the portability of your applications. The information contained in the System Variables is often similar, though not always exactly the same as that returned by the functions. If you are replacing these functions in your program to favor the use of System Variables, be sure to understand the differences in the information offered.

The following function descriptions indicate when a System Variable is available for similar information.

What Is in This Section?

This section contains the following topics:

Topic	Page
GetBatteryLevel: Returns Remaining Power Charge of the Battery	27
GetEventsNumber: Returns the Number of External Events Detected	28
GetFirmwareVersion: Returns Information About the Firmware, the Boot and the Coprocessor Versions	29
GetHardwareVersion: Returns the Hardware Version	30
GetLastStopCause: Returns the Cause Last Stop	31
GetLastStopTime: Returns the Date and Time of the Last Detected Stop	32
GetLocalIOStatus: Returns the Embedded I/O Status	33
GetPlcFault: Returns Detected Errors on the Controller I/O	35
GetRightBusStatus: Returns the Status of the Expansion Bus	36
GetSerialNumber: Returns the Serial Number of the Controller	40
GetShortCutStatus: Returns the Short-Circuit Status on Embedded Outputs	41
IsFirstMastColdCycle: Indicates if Cycle is the First MAST Cold Start Cycle	42
IsFirstMastCycle: Indicates if Cycle is the First MAST Cycle	43
IsFirstMastWarmCycle: Indicates if Cycle is the First MAST Warm Start Cycle	45

GetBatteryLevel: Returns Remaining Power Charge of the Battery

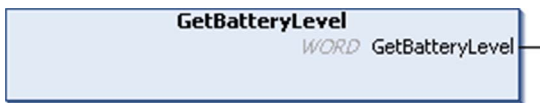
Function Description

This function returns the remaining power charge of the external backup battery (in percent).

For more information about internal and external batteries, refer to the M238 Hardware Guide (see *M238 Logic Controller, Hardware Guide*).

NOTE: This information is also available through the System Variable PLC_R.i_wBatteryStatus (see *page 18*)

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 73*).

I/O Variables Description

The table describes the output variable:

Output	Type	Comment
GetBatteryLevel	WORD	Percentage of charge remaining in the battery. Range: 0...100: <ul style="list-style-type: none"> ● 100%= 3.5 V ● 50% = 2.75 V ● 0%= 2 V Flashing thresholds of the Batt LED: <ul style="list-style-type: none"> ● 100%= always OFF ● 50%= periodic flashing ● 0%= always ON

GetEventsNumber: Returns the Number of External Events Detected

Function Description

This function returns the number of events that have occurred since the last cold start, including those detected on inputs and HSC threshold compare events.

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 73).

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
GetEventsNumber	DWORD	The value representing the number of events that have occurred since the last cold start. Reset to 0 with a call of the function ResetEventsNumber (see page 48).

GetFirmwareVersion: Returns Information About the Firmware, the Boot and the Coprocessor Versions

Function Description

This function returns the following information about the Controller:

- Firmware version
- Boot version (Kernel for firmware update)
- Coprocessor version (ASIC dedicated to embedded I/O management)

NOTE: The similar information are also available through the System Variables

PLC_R.i_byFirmVersion, PLC_R.i_byBootVersion and PLC_R.i_byChipVersion
(see page 18)

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 73).

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
GetFirmwareVersion	FIRMWARE_VERSION (see page 61)	Firmware, Boot and Coprocessor version.

Example

This example describes how to use the structured variable to get the versions values:

```

VAR
    MyFirmwareVersion : FIRMWARE_VERSION;
    MyFwVersion       : DWORD;
    MyBootVersion     : WORD;
    MyCoProcVersion  : WORD;
END_VAR
MyFirmwareVersion := GetFirmwareVersion();
MyFwVersion       := MyFirmwareVersion.FwVersion;
MyBootVersion     := MyFirmwareVersion.BootVersion;
MyCoProcVersion  := MyFirmwareVersion.AsicVersion;

```

GetHardwareVersion: Returns the Hardware Version

Function Description

This function returns the hardware version of the Controller.

NOTE: The similar information are also available through the System Variables

PLC_R.i_dwHardVersion ([see page 18](#))

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 73](#)).

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
GetHardwareVersion	WORD	Hardware version of the Controller.

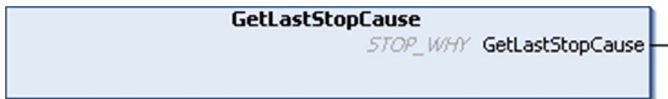
GetLastStopCause: Returns the Cause Last Stop

Function Description

This function returns the cause of the last transition from RUN to another state.

NOTE: The similar information value is also available through the System Variable PLC_R.i_wLastStopCause ([see page 18](#)).

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 73](#)).

I/O Variable Description

The following table describes the output variable:

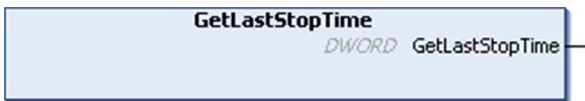
Output	Type	Comment
GetLastStopCause	STOP_WHY (see page 63)	Cause of the last transition from RUN to another state.

GetLastStopTime: Returns the Date and Time of the Last Detected Stop

Function Description

This function returns the date and time of the last transition from RUN to another state.

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 73).

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
GetLastStopTime	DWORD	The time of the last detected STOP in seconds beginning with January 1, 1970 at 00:00.

GetLocalIOStatus: Returns the Embedded I/O Status

Function Description

This function returns the embedded I/O status.

NOTE: The similar information is also available through the System Variable `PLC_R.i_wLocalIOStatus` (*see page 18*).

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 73*).

I/O Variables Description

The table describes the input parameter:

Input	Type	Comment
Mode	LOCAL_IO_GET_STATUS (<i>see page 64</i>)	Parameter of the function: currently only LOCAL_IO_GET_GEN_STATUS (00 hex) (<i>see page 64</i>) is available.

The table describes the output variable:

Output	Type	Comment
GetLocalIOStatus	LOCAL_IO_GEN_STATUS (<i>see page 65</i>)	Status of the embedded I/O.

Example 1

This example shows a direct use of `LOCAL_IO_GET_GEN_STATUS` enumerator of the `LOCAL_IO_GET_STATUS` enumeration type for the Mode input parameter:

```
VAR
    MyLocalIOStatus : LOCAL_IO_GEN_STATUS;
END_VAR

MyLocalIOStatus := GetLocalIOStatus(LOCAL_IO_GET_GEN_STATUS);
```

Example 2

This example shows the use of an intermediate variable for Mode input parameter.

```
VAR
    MyLocalIOStatus : LOCAL_IO_GEN_STATUS;
    MyMode : LOCAL_IO_GET_STATUS;
END_VAR

MyMode := LOCAL_IO_GET_GEN_STATUS;
MyLocalIOStatus := GetLocalIOStatus(MyMode);
```

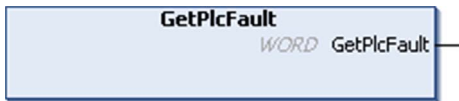
GetPlcFault: Returns Detected Errors on the Controller I/O

Function Description

This function returns a generic diagnostic of the Controller I/O in a bit field (FFFF hex means no error detected):

NOTE: The similar information is also available through the System Variable (bit0 and bit1) PLC_R.i_lwSystemFault_1 ([see page 18](#)).

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 73](#)).

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
GetPlcFault	WORD	Diagnostic of controller's I/O in a bit field (FFFF hex means no error detected).

The following table details the bits contained in the output bit field:

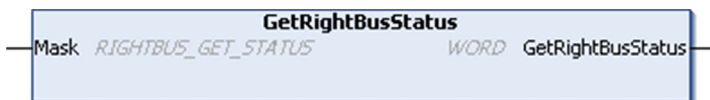
Bit	Meaning
0	FALSE = detected error on the I/O expansion bus. Use the <code>GetRightBusStatus</code> (see page 36) function to get detailed diagnostic information.
1	FALSE = detected error on the embedded I/O. Use the <code>GetLocalIOStatus</code> (see page 33) function to get detailed diagnostic information.
2..15	Reserved, always 1

GetRightBusStatus: Returns the Status of the Expansion Bus

Function Description

This function returns the I/O Expansion Bus status in a bit field. Following the input parameter (`Mask`), the function returns a configuration diagnostic on the I/O expansion bus (I/O expansion bus configuration mismatch with plugged modules) or a detailed diagnostic of the requested expansion analog I/O module.

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 73).

I/O Variables Description

The following table describes the input parameter:

Input	Type	Comment
Mask	RIGHTBUS_GET_STATUS (see page 66)	Defines the type of I/O expansion bus diagnostic returned by the function: bus configuration or one of the 7 possible expansion modules.

The following table describes the output variable:

Output	Type	Comment
GetRightBusStatus	WORD	I/O expansion bus diagnostic. see details below

I/O Expansion Bus Configuration Diagnostic

The table below describes the bit field returned by the function `GetRightBusStatus` when the input parameter (`Mask`) is `RIGHT_BUS_GET_GEN_STATUS` (see page 66) for a configuration diagnostic of the expansion bus (0000 hex if no detected error):

Bit	Description
0	Reserved (always 0)
1	TRUE if the TM2 module 1 configuration mismatches with plugged module.
2	TRUE if the TM2 module 2 configuration mismatches with plugged module.
3	TRUE if the TM2 module 3 configuration mismatches with plugged module.
4	TRUE if the TM2 module 4 configuration mismatches with plugged module.
5	TRUE if the TM2 module 5 configuration mismatches with plugged module.
6	TRUE if the TM2 module 6 configuration mismatches with plugged module.
7	TRUE if the TM2 module 7 configuration mismatches with plugged module.
8	TRUE if the right bus is blocked.
7...15	Reserved (always 0)

TM2 Analog I/O Expansion Module Diagnostic

The tables below describe the bit field returned by the function `GetRightBusStatus` when the input parameter (`Mask`) is `RIGHTBUS_GET_STATUSx` (see page 66) (where `x=1 to 7`) for a detailed diagnostic of the expansion module "x".

NOTE: The detailed diagnostic is valid for analog I/O modules only and the bit field meaning depends on the type of the concerned analog module.

When the associated module is a standard analog I/O module (up to 2 input channels):

- TM2AMM3HT
- TM2ALM3LT
- TM2AMI2HT
- TM2AMI2LT
- TM2AVO2HT
- TM2AMO1HT

Bit	Description
0	All analog channels in normal state
1	Module in initialization state
2	Power supply not operating correctly
3	Incorrect configuration - analysis needed
4	Conversion process for input channel 0
5	Conversion process for input channel 1
6	Invalid parameter for input channel 0

Bit	Description
7	Invalid parameter for input channel 1
8	Not used
9	Not used
10	Overflow value for input channel 0
11	Overflow value for input channel 1
12	Underflow value for input channel 0
13	Underflow value for input channel 1
14	Not used
15	Invalid parameter for output channel

When the associated module is one of the 4 or 8 analog input channels modules:

- TM2ARI8HT
- TM2AMI8HT
- TM2ARI8LT
- TM2ARI8LRJ
- TM2AMI4LT
- TM2AMM6HT

Bit	Description	Meaning
0, 1	Channel 0 state	00: Analog channel in normal state 01: Invalid parameter for input channel 10: Unavailable input value (module in initialization state, conversion process) 11: Invalid value for input channel (overflow or underflow value)
2, 3	Channel 1 state	see bit 0, 1
4, 5	Channel 2 state	see bit 0, 1
6, 7	Channel 3 state	see bit 0, 1
8, 9	Channel 4 state	see bit 0, 1 (for 8 input channels modules only)
10, 11	Channel 5 state	see bit 0, 1 (for 8 input channels modules only)
12, 13	Channel 6 state	see bit 0, 1 (for 8 input channels modules only)
14, 15	Channel 7 state	see bit 0, 1 (for 8 input channels modules only)

NOTE: When the targeted expansion module is a digital I/O module, a High Speed Counter module or an AS-Interface Master Module, the diagnostic returned is invalid (0000 hex).

To obtain a diagnostic on HSC and AS-Interface Expansion modules, you can use:

- HSCGetDiag (see *Modicon M238 Logic Controller, High Speed Counting, M238 HSC Library Guide*) **Function Block for the TM200 HSC206D• module**
- ASI_MasterStatusCheck (see *Modicon M238 Logic Controller, Programming Guide*) **for the TWDNOI10M3 module**

Example

The following example describes a method using GetRightBusStatus for I/O expansion bus and analog modules diagnostic:

```

VAR
  (*Modules 1 to 7 conf diag = MyRightBusStatus bits 1 to 7*)
  MyRightBusStatus: WORD;
  (*Modules 1 to 7 diag codes*)
  ModuleError:Array [1..7] of WORD;
END_VAR

(*Conf diagnostic on expansion bus*)
MyRightBusStatus:=GetRightBusStatus
(RIGHTBUS_GET_GEN_STATUS);

IF MyRightBusStatus<>0 THEN
  (*Conf mismatch detected => set an alarm, check bit values...*)
  END_IF;

(*Get modules diagnostic; detected error if diag <> 0*)
(*Limit the list to analog modules configured only*)
ModuleError[1]:=GetRightBusStatus(RIGHTBUS_GET_STATUS1);
ModuleError[2]:=GetRightBusStatus(RIGHTBUS_GET_STATUS2);
ModuleError[3]:=GetRightBusStatus(RIGHTBUS_GET_STATUS3);
ModuleError[4]:=GetRightBusStatus(RIGHTBUS_GET_STATUS4);
ModuleError[5]:=GetRightBusStatus(RIGHTBUS_GET_STATUS5);
ModuleError[6]:=GetRightBusStatus(RIGHTBUS_GET_STATUS6);
ModuleError[7]:=GetRightBusStatus(RIGHTBUS_GET_STATUS7);

```

GetSerialNumber: Returns the Serial Number of the Controller

Function Description

This function returns the serial number of the Controller.

NOTE: The similar information is also available through the System Variable (bit0 and bit1) `PLC_R.i_dwSerialNumber` ([see page 18](#)).

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 73](#)).

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
GetSerialNumber	DWORD	Controller Serial number.

GetShortCutStatus: Returns the Short-Circuit Status on Embedded Outputs

Function Description

This function returns the short-circuit or overload diagnostic on embedded outputs.

NOTE: For more information about embedded outputs management, refer to the M238 Hardware Guide (see *M238 Logic Controller, Hardware Guide*).

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 73](#)).

I/O Variable Description

The table describes the output variable:

Parameter	Type	Comment
GetShortCutStatus	WORD	see bit field description below

The following table describes the bit field for the TM238LDD24DT and TM238LFDC24DT:

Bit	Description
0	TRUE = short-circuit on outputs group 1 (Q0 to Q3).
1	TRUE = short-circuit on outputs group 2 (Q4 to Q6).
2	TRUE = short-circuit on outputs group 3 (Q7 to Q9).

The following table describes the bit field for the TM238LDA24DR and TM238LFAC24DR:

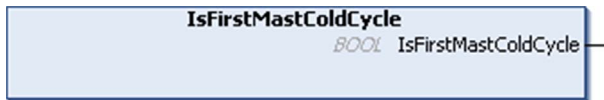
Bit	Description
0	TRUE = short-circuit on outputs category 1 (Q0 to Q3).

IsFirstMastColdCycle: Indicates if Cycle is the First MAST Cold Start Cycle

Function Description

This function returns TRUE during the first MAST cycle after a cold start (first cycle after download or reset cold).

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 73).

I/O Variable Description

The table describes the output variable:

Output	Type	Comment
IsFirstMastColdCycle	BOOL	TRUE during the first MAST task cycle after a cold start.

Example

Refer to the function `IsFirstMastCycle` (see page 43).

IsFirstMastCycle: Indicates if Cycle is the First MAST Cycle

Function Description

This function returns TRUE during the first MAST cycle after a start.

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 73).

I/O Variable Description

Output	Type	Comment
IsFirstMastCycle	BOOL	TRUE during the first MAST task cycle after a start.

Example

This example describes the three functions `IsFirstMastCycle`, `IsFirstMastColdCycle` and `IsFirstMastWarmCycle` used together.

Use this example in MAST task. Otherwise, it may run several times or possibly never (an additional task might be called several times or not called during 1 MAST task cycle):

```

VAR
MyIsFirstMastCycle : BOOL;
MyIsFirstMastWarmCycle : BOOL;
MyIsFirstMastColdCycle : BOOL;
END_VAR

MyIsFirstMastWarmCycle := IsFirstMastWarmCycle();
MyIsFirstMastColdCycle := IsFirstMastColdCycle();
MyIsFirstMastCycle := IsFirstMastCycle();

IF (MyIsFirstMastWarmCycle) THEN

(*This is the first Mast Cycle after a Warm Start: all variables are set
to their initialization values except the Retain variables*)

(*=> initialize the needed variables so that your application runs as
expected in this case*)

END_IF;
```

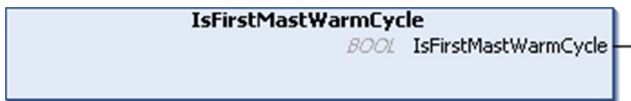
```
IF (MyIsFirstMastColdCycle) THEN
(*This is the first Mast Cycle after a Cold Start: all variables are set
to their initialization values including the Retain Variables*)
(*=> initialize the needed variables so that your application runs as
expected in this case*)
END_IF;
IF (MyIsFirstMastCycle) THEN
(*This is the first Mast Cycle after a Start, i.e. after a Warm or Cold
Start as well as STOP/RUN commands*)
(*=> initialize the needed variables so that your application runs as
expected in this case*)
END_IF;
```

IsFirstMastWarmCycle: Indicates if Cycle is the First MAST Warm Start Cycle

Function Description

This function returns TRUE during the first MAST cycle after a warm start.

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 73).

I/O Variable Description

This table describes the output variable:

Output	Type	Comment
IsFirstMastWarmCycle	BOOL	TRUE during the first MAST task cycle after a warm start.

Example

Refer to the function IsFirstMastCycle (see page 43).

Section 2.2

M238 Write Functions

Overview

This section describes the write functions included in the M238 PLCSystem library.

In many cases, similar information returned by these functions can also be obtained through System Variables (*see page 13*). System Variables are common across the control platforms, and therefore it is recommended to use them where possible for the portability of your applications. The information contained in the System Variables is often similar, though not always exactly the same as that returned by the functions. If you are replacing these functions in your program to favor the use of System Variables, be sure to understand the differences in the information offered.

The following function descriptions indicate when a System Variable is available for similar information.

What Is in This Section?

This section contains the following topics:

Topic	Page
InhibitBatLowLed: Disables or Re-enables the Batt LED	47
ResetEventsNumber: Resets Events Number	48
SetRTCDrift: Adjust the Real Time Clock Each Week	49

InhibitBatLowLed: Disables or Re-enables the Batt LED

Function Description

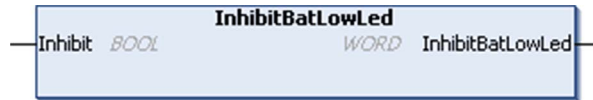
This function disables or re-enables the display of the low battery LED indicator (Batt).

It may be used when the controller is used without external battery, to avoid having a persistent error signal. The display of the low battery LED indicator (Batt) is automatically restored after a **Reset to Origin** command or a download from the programming panel to the controller.

NOTE: For more information about the external battery, refer to the M238 Hardware Guide (see *M238 Logic Controller, Hardware Guide*).

NOTE: For more information about the LED indicator, refer to the M238 Hardware Guide (see *M238 Logic Controller, Hardware Guide*).

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 73).

I/O Variable Description

The following table describes the input parameter:

Input	Type	Comment
Inhibit	BOOL	TRUE = Disables the Display of the Batt LED. FALSE = Re-enables the display of the Batt LED.

The following table describes the output parameter:

Output	Type	Comment
InhibitBatLowLed	WORD	00 hex = operation successful else = operation unsuccessful

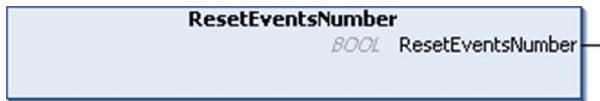
ResetEventsNumber: Resets Events Number

Function Description

This function resets the number of events that have occurred since the last cold start, including those detected on inputs and HSC threshold compare events.

NOTE: The number of events detected on inputs or HSC threshold detection is returned by the function `GetEventsNumber` ([see page 28](#)).

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 73](#)).

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
ResetEventsNumber	BOOL	TRUE if the counter has been reset to 0 with success.

SetRTCDrift: Adjust the Real Time Clock Each Week

Function Description

This function adds or subtracts to the Real Time Clock (RTC) a specified number of seconds, each week, at the specified day of the week and time (hour:minute).

NOTE: The SetRTCDrift function must be called at least once to update the parameters.

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 73).

I/O Variables Description

This table describes the input parameters:

Inputs	Type	Comment
RtcDrift	SINT (-59..59)	Correction in seconds (-59 ... +59)
Day	DAY_OF_WEEK (see page 67)	Day of change.
Hour	HOUR (see page 68)	Hour of change.
Minute	MINUTE (see page 69)	Minute of change.

NOTE: If RTCDrift > 0, the execution of the change takes place at Hour:Minute:00 seconds. If RTCDrift < 0, the execution of the change takes place at Hour:Minute:59 seconds.

NOTE: If the values entered for RtcDrift, Day, Hour, Minute exceed the limit values, the logic controller firmware sets all values to their maximum values.

This table describes the output variable:

Output	Type	Comment
SetRTCDrift	WORD	Returns 00 hex if command is correct otherwise returns the ID code of the detected error.

Example

In this example, the function is called only once during the first MAST task cycle and 20 seconds is added to the RTC every Tuesday at 5:45:00 a.m.:

```
VAR
    MyRTCDrift : SINT (-59..59) := 0;
    MyDay : DAY_OF_WEEK;
    MyHour : HOUR;
    MyMinute : MINUTE;
END_VAR

IF IsFirstMastCycle() THEN
    MyRTCDrift := 20;
    MyDay := TUESDAY;
    MyHour := 5;
    MyMinute := 45;
    SetRTCDrift(MyRTCDrift, MyDay, MyHour, MyMinute);
END_IF
```

Chapter 3

M238 PLCSystem Library Data Types

Overview

This chapter describes the data types of the M238 PLCSystem Library.

There are 2 kinds of data types available:

- System variable data types are used by the system variables (*see page 11*) of the M238 PLCSystem Library (PLC_R, PLC_W,...).
- System function data types are used by the read/write system functions (*see page 25*) of the M238 PLCSystem Library.

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
3.1	PLC_R/W System Variables Data Types	52
3.2	System Function Data Types	60

Section 3.1

PLC_R/W System Variables Data Types

Overview

This section lists and describes the system variable data types included in the `PLC_R` and `PLC_W` structures.

What Is in This Section?

This section contains the following topics:

Topic	Page
PLC_R_STATUS: Controller Status Codes	53
PLC_R_APPLICATION_ERROR: Detected Application Error Status Codes	54
PLC_R_BOOT_PROJECT_STATUS: Boot Project Status Codes	56
PLC_R_IO_STATUS: I/O Status Codes	57
PLC_R_STOP_CAUSE: From RUN State to Other State Transition Cause Codes	58
PLC_W_COMMAND: Control Command Codes	59

PLC_R_STATUS: Controller Status Codes

Enumerated Type Description

The PLC_R_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
PLC_R_EMPTY	0000 hex	Controller does not contain an application.
PLC_R_STOPPED	0001 hex	Controller is stopped.
PLC_R_RUNNING	0002 hex	Controller is running.
PLC_R_HALT	0004 hex	Controller is in a HALT state. (see the controller state diagram in your controller <i>programming guide</i>).
PLC_R_BREAKPOINT	0008 hex	Controller has paused at a breakpoint.

PLC_R_APPLICATION_ERROR: Detected Application Error Status Codes

Enumerated Type Description

The PLC_R_APPLICATION_ERROR enumeration data type contains the following values:

Enumerator	Value	Comment	What to do
PLC_R_APP_ERR_UNKNOWN	FFFF hex	Undefined error detected.	Contact your local support.
PLC_R_APP_ERR_NOEXCEPTION	0000 hex	No error detected.	–
PLC_R_APP_ERR_WATCHDOG	0010 hex	Task watchdog expired.	Check your application. See chapter . A reset is needed to enter Run mode.
PLC_R_APP_ERR_HARDWAREWATCHDOG	0011 hex	System watchdog expired.	If the problem is reproducible, check for disconnected communication ports. If the problem persists, update the firmware. If the problem still persists, contact your local support.
PLC_R_APP_ERR_IO_CONFIG_ERROR	0012 hex	Incorrect I/O configuration parameters detected.	Your application might be corrupted. To resolve this issue, use one of the methods: <ol style="list-style-type: none"> 1. Run → Build → Clean All 2. Export/Import your application. 3. Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_UNRESOLVED_EXTREFS	0018 hex	Undefined functions detected.	Delete the unresolved functions from the application.
PLC_R_APP_ERR_IEC_TASK_CONFIG_ERROR	0025 hex	Incorrect Task configuration parameters detected.	Your application might be corrupted. To resolve this issue, use one of the methods: <ol style="list-style-type: none"> 1. Run → Build → Clean All 2. Export/Import your application. 3. Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_ILLEGAL_INSTRUCTION	0050 hex	Undefined instruction detected.	Debug your application to resolve the problem.

Enumerator	Value	Comment	What to do
PLC_R_APP_ERR_ACCESS_VIOLATION	0051 hex	Attempted access to reserved memory area.	Debug your application to resolve the problem.
PLC_R_APP_ERR_DIVIDE_BY_ZERO	0102 hex	Integer division by zero detected.	Debug your application to resolve the problem.
PLC_R_APP_ERR_PROCESSORLOAD_WATCHDOG	0105 hex	Processor overloaded by Application Tasks.	Reduce the application workload by improving the application architecture. Increase the task cycle duration. Reduce event frequency.
PLC_R_APP_ERR_DIVIDE_REAL_BY_ZERO	0152 hex	Real division by zero detected.	Debug your application to resolve the problem.

PLC_R_BOOT_PROJECT_STATUS: Boot Project Status Codes

Enumerated Type Description

The PLC_R_BOOT_PROJECT_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
PLC_R_NO_BOOT_PROJECT	0000 hex	Not used
PLC_R_BOOT_PROJECT_CREATION_IN_PROGRESS	0001 hex	Boot project is being created.
PLC_R_DIFFERENT_BOOT_PROJECT	0002 hex	Boot project in Flash is different from the project loaded in RAM or does not exist.
PLC_R_VALID_BOOT_PROJECT	FFFF hex	Boot project in Flash is the same as the project loaded in RAM.

PLC_R_IO_STATUS: I/O Status Codes

Enumerated Type Description

The PLC_R_IO_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
PLC_R_IO_OK	FFFF hex	Inputs/Outputs are operational.
PLC_R_IO_NO_INIT	0001 hex	Inputs/Outputs are not initialized.
PLC_R_IO_CONF_FAULT	0002 hex	Incorrect I/O configuration parameters detected.
PLC_R_IO_SHORTCUT_FAULT	0003 hex	Inputs/Outputs short-circuit detected.
RESERVED	0004 hex	Not used
PLC_R_IO_COM_LOST	0005 hex	Communication error detected with Coprocessor. The embedded I/O are not updated.

PLC_R_STOP_CAUSE: From RUN State to Other State Transition Cause Codes

Enumerated Type Description

The PLC_R_STOP_CAUSE enumeration data type contains the following values:

Enumerator	Value	Comment	What to do
PLC_R_STOP_REASON_UNKNOWN	00 hex	Initial value or stop cause is undefined.	Contact your local support in case of undefined stop cause.
PLC_R_STOP_REASON_HW_WATCHDOG	01 hex	Stopped after hardware watchdog timeout.	Contact your local support.
PLC_R_STOP_REASON_RESET	02 hex	Stopped after reset.	See reset possibilities in chapter .
PLC_R_STOP_REASON_EXCEPTION	03 hex	Stopped after exception.	Check your application. See chapter . A reset is needed to enter Run mode.
PLC_R_STOP_REASON_USER	04 hex	Stopped after a user request.	See chapter .
PLC_R_STOP_REASON_IECPROGRAM	05 hex	Stopped after a program command request (for example: control command with parameter PLC_W.q_wPLCControl: =PLC_W_COMMAND.PL_C_W_STOP;).	–
PLC_R_STOP_REASON_DELETE	06 hex	Stopped after a remove application command.	See chapter .
PLC_R_STOP_REASON_DEBUGGING	07 hex	Stopped after entering debug mode.	–
PLC_R_STOP_REASON_NETWORK_REQUEST	0A hex	Stopped after a request from the network, USB key, or PLC_W command.	–
PLC_R_STOP_REASON_INPUT	0B hex	Stop required by a controller input.	–
PLC_R_STOP_REASON_RETAIN_MISMATCH	0C hex	Stopped after an unsuccessful check context test during rebooting.	Retain variables were deleted because they were not referenced in the application. If the application sets retain variables to their initialization values, the is available.
PLC_R_STOP_REASON_BOOT_APPLI_MISMATCH	0D hex	Stopped after an unsuccessful compare between the boot application and the application that was in the memory before rebooting.	Create a valid boot application.
PLC_R_STOP_REASON_POWERFAIL	0E hex	Stopped after a power interruption.	–

PLC_W_COMMAND: Control Command Codes

Enumerated Type Description

The PLC_W_COMMAND enumeration data type contains the following values:

Enumerator	Value	Comment
PLC_W_STOP	0001 hex	Command to stop the controller.
PLC_W_RUN	0002 hex	Command to run the controller.
PLC_W_RESET_COLD	0004 hex	Command to initiate a Controller cold reset.
PLC_W_RESET_WARM	0008 hex	Command to initiate a Controller warm reset.

Section 3.2

System Function Data Types

Overview

This section describes the different system function data types of the M238 PLCSystem library.

What Is in This Section?

This section contains the following topics:

Topic	Page
FIRMWARE_VERSION: GetFirmwareVersion Function Output Type	61
BOOT_PROJECT_STATUS: GetBootProjectStatus Function Output Codes	62
STOP_WHY: GetLastStopCause Function Output Codes	63
LOCAL_IO_GET_STATUS: GetLocalIOStatus Function Parameter Codes	64
LOCAL_IO_GEN_STATUS: GetLocalIOStatus Function Output Codes	65
RIGHTBUS_GET_STATUS: GetRightBusStatus Function Parameter Codes	66
DAY_OF_WEEK: SetRTCDrift Function Day Parameter Codes	67
HOUR: SetRTCDrift Function Hour Parameter Type	68
MINUTE: SetRTCDrift Function Minute Parameter Type	69

FIRMWARE_VERSION: GetFirmwareVersion Function Output Type

Structure Description

The data structure contains the following variables:

Variable	Type	Description
FwVersion	DWORD	Contains the firmware version (for example, 2.0.20.4 = 02001404 hex).
BootVersion	WORD	Contains the boot version (for example, 1.4 = 0104 hex).
AsicVersion	WORD	Contains the processor version.

BOOT_PROJECT_STATUS: GetBootProjectStatus Function Output Codes

Enumerated Type Description

The enumeration data type contains the following values:

Enumerator	Value	Description
VALID_BOOT_PROJECT	0000 hex	Boot project in Flash is the same as the project loaded in RAM.
INVALID_BOOT_PROJECT	0001 hex	Boot project in Flash is different from the project loaded in RAM or a boot project does not exist.
CREATE_BOOT_PROJECT_IN_PROGRESS	0002 hex	Boot project is being created.

STOP_WHY: GetLastStopCause Function Output Codes

Enumerated Type Description

The enumeration data type contains the status with the following values:

Enumerator	Value	Description
STOP_FROM_POWER_FAIL	00 hex	Stop after a power outage.
STOP_FROM_INTERNAL_ERROR	01 hex	Stop due to internal error detection.
STOP_FROM_APP_WATCHDOG	02 hex	Stop after a Task or a System Watchdog (see <i>Modicon M238 Logic Controller, Programming Guide</i>).
STOP_FROM_STOP_REQUEST	03 hex	Stop after a user request.
STOP_FROM_POWER_FAIL_BAD_RETAINS	04 hex	The Retain Variables have been lost after a power outage. This may happen if the internal battery has no charge and external battery has no charge or is not present.
STOP_FROM_POWER_FAIL_INVALID_BOOT_PROJECT	05 hex	The Boot Application project not valid at power off because of an online change without updating the boot application.

LOCAL_IO_GET_STATUS: GetLocalIOStatus Function Parameter Codes

Enumerated Type Description

The LOCAL_IO_GET_STATUS enumeration data type contains the following value:

Enumerator	Value	Description
LOCAL_IO_GET_GEN_STATUS	00 hex	Value used to request the general status of embedded I/Os.

LOCAL_IO_GEN_STATUS: GetLocalIOStatus Function Output Codes

Enumerated Type Description

The LOCAL_IO_GEN_STATUS enumeration data type contains the status of local I/Os with the following values:

Enumerator	Value	Description
LOCAL_IO_OK	00 hex	Inputs / Outputs are operational.
LOCAL_IO_NO_INIT	01 hex	Incorrect I/O configuration parameters detected.
LOCAL_IO_COM_LOST	02 hex	I/O communication error detected. The embedded I/O are not updated.
LOCAL_IO_CONF_FAULT	03 hex	Incorrect I/O configuration parameters detected.

RIGHTBUS_GET_STATUS: GetRightBusStatus Function Parameter Codes

Enumerated Type Description

The enumeration data type contains the following values:

Enumerator	Value	Description
RIGHTBUS_GET_GEN_STATUS	00 hex	Parameter for expansion bus configuration diagnostic
RIGHTBUS_GET_STATUS1	01 hex	Parameter for expansion bus Module 1 diagnostic
RIGHTBUS_GET_STATUS2	02 hex	Parameter for expansion bus Module 2 diagnostic
RIGHTBUS_GET_STATUS3	03 hex	Parameter for expansion bus Module 3 diagnostic
RIGHTBUS_GET_STATUS4	04 hex	Parameter for expansion bus Module 4 diagnostic
RIGHTBUS_GET_STATUS5	05 hex	Parameter for expansion bus Module 5 diagnostic
RIGHTBUS_GET_STATUS6	06 hex	Parameter for expansion bus Module 6 diagnostic
RIGHTBUS_GET_STATUS7	07 hex	Parameter for expansion bus Module 7 diagnostic

NOTE: For more information on using the `RIGHTBUS_GET_STATUS` parameter type, refer to the function `GetRightBusStatus` ([see page 36](#)).

DAY_OF_WEEK: SetRTCDrift Function Day Parameter Codes

Enumerated Type Description

The enumeration data type contains the following values:

Enumerator	Value	Comment
MONDAY	01 hex	Set the day of week to Monday
TUESDAY	02 hex	Set the day of week to Tuesday
WEDNESDAY	03 hex	Set the day of week to Wednesday
THURSDAY	04 hex	Set the day of week to Thursday
FRIDAY	05 hex	Set the day of week to Friday
SATURDAY	06 hex	Set the day of week to Saturday
SUNDAY	07 hex	Set the day of week to Sunday

HOURL: SetRTCDrift Function Hour Parameter Type

Data Type Description

The data type contains the hour values from 0 to 23.

MINUTE: SetRTCDrift Function Minute Parameter Type

Data Type Description

The data type contains the minute values from 0 to 59.

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	74
How to Use a Function or a Function Block in IL Language	75
How to Use a Function or a Function Block in ST Language	79

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:

```
1 PROGRAM MyProgram_ST
2 VAR
3     Timer_ON: TON; // Function Block Instance
4     Timer_RunCd: BOOL;
5     Timer_PresetValue: TIME := T#5S;
6     Timer_Output: BOOL;
7     Timer_ElapsedTime: TIME;
8 END_VAR
```

```
1 Timer_ON(
2     IN:=Timer_RunCd,
3     PT:=Timer_PresetValue,
4     Q=>Timer_Output,
5     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 <i>Adding and Calling POU's (see SoMachine, Programming Guide)</i> .
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: <ul style="list-style-type: none"> 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: <code>IsFirstMastCycle</code>	
with input parameters: <code>SetRTCDrift</code>	

In IL language, the function name is used directly in the operator column:

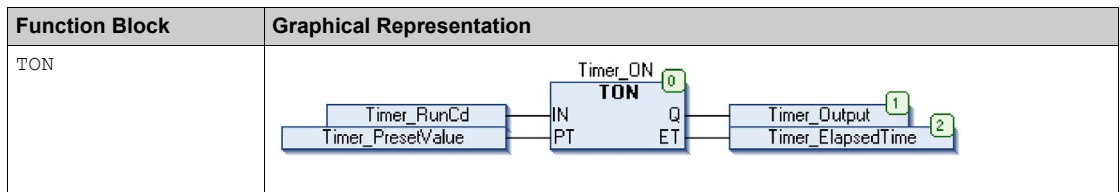
Function	Representation in SoMachine POU IL Editor															
IL example of a function without input parameter: IsFirstMastCycle	<pre> 1 PROGRAM MyProgram_IL 2 VAR 3 FirstCycle: BOOL; 4 END_VAR </pre> <hr/> <table border="1" data-bbox="371 459 979 570"> <tr> <td data-bbox="371 459 444 488">1</td> <td data-bbox="444 459 742 488">IsFirstMastCycle</td> <td data-bbox="742 459 979 488"></td> </tr> <tr> <td data-bbox="371 488 444 519"></td> <td data-bbox="444 488 742 519">ST</td> <td data-bbox="742 488 979 519">FirstCycle</td> </tr> <tr> <td data-bbox="371 519 444 552"></td> <td data-bbox="444 519 742 552"></td> <td data-bbox="742 519 979 552"></td> </tr> </table>	1	IsFirstMastCycle			ST	FirstCycle									
1	IsFirstMastCycle															
	ST	FirstCycle														
IL example of a function with input parameters: SetRTCDrift	<pre> 1 PROGRAM MyProgram_IL 2 VAR 3 myDrift: SINT (-29..29) := 5; 4 myDay: DAY_OF_WEEK := SUNDAY; 5 myHour: HOUR := 12; 6 myMinute: MINUTE; 7 myDiag: RTCSETDRIFT_ERROR; 8 END_VAR </pre> <hr/> <table border="1" data-bbox="371 971 930 1149"> <tr> <td data-bbox="371 971 444 1002">1</td> <td data-bbox="444 971 687 1002">LD</td> <td data-bbox="687 971 930 1002">myDrift</td> </tr> <tr> <td data-bbox="371 1002 444 1032"></td> <td data-bbox="444 1002 687 1032">SetRTCDrift</td> <td data-bbox="687 1002 930 1032">myDay</td> </tr> <tr> <td data-bbox="371 1032 444 1063"></td> <td data-bbox="444 1032 687 1063"></td> <td data-bbox="687 1032 930 1063">myHour</td> </tr> <tr> <td data-bbox="371 1063 444 1094"></td> <td data-bbox="444 1063 687 1094"></td> <td data-bbox="687 1063 930 1094">myMinute</td> </tr> <tr> <td data-bbox="371 1094 444 1125"></td> <td data-bbox="444 1094 687 1125">ST</td> <td data-bbox="687 1094 930 1125">myDiag</td> </tr> </table>	1	LD	myDrift		SetRTCDrift	myDay			myHour			myMinute		ST	myDiag
1	LD	myDrift														
	SetRTCDrift	myDay														
		myHour														
		myMinute														
	ST	myDiag														

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 POU's (see <i>SoMachine, Programming Guide</i>).
2	Create the variables that the function block requires, including the instance name.
3	Function Blocks are called using a <code>CAL</code> instruction: <ul style="list-style-type: none"> ● Use the Input Assistant to select the FB (right-click and select Insert Box in the context menu). ● Automatically, the <code>CAL</code> instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: <ul style="list-style-type: none"> ● Values to inputs are set by " := ". ● Values to outputs are set by " => ".
4	In the <code>CAL</code> 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:



In IL language, the function block name is used directly in the operator column:

Function Block	Representation in SoMachine POU IL Editor
TON	<pre> 1 PROGRAM MyProgram_IL 2 VAR 3 Timer_ON: TON; // Function Block instance declaration 4 Timer_RunCd: BOOL; 5 Timer_PresetValue: TIME := T#5S; 6 Timer_Output: BOOL; 7 Timer_ElapsedTime: TIME; 8 END_VAR 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 </pre>

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.

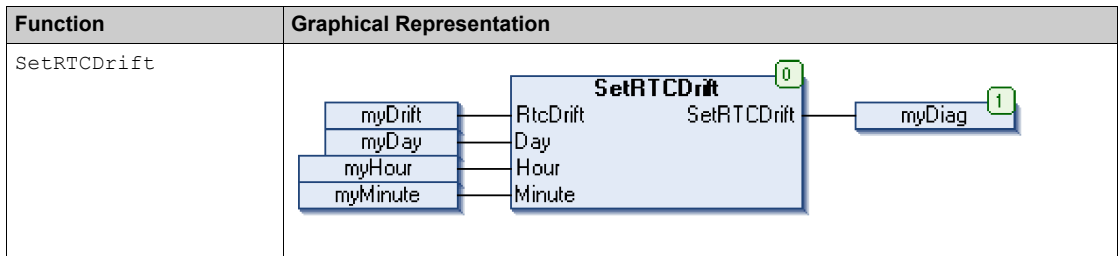
Function `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

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 <i>SoMachine, Programming Guide</i>).
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: <code>FunctionResult:= FunctionName(VarInput1, VarInput2,.. VarInputx);</code>

To illustrate the procedure, consider the function `SetRTCDrift` graphically presented below:



The ST language of this function is the following:

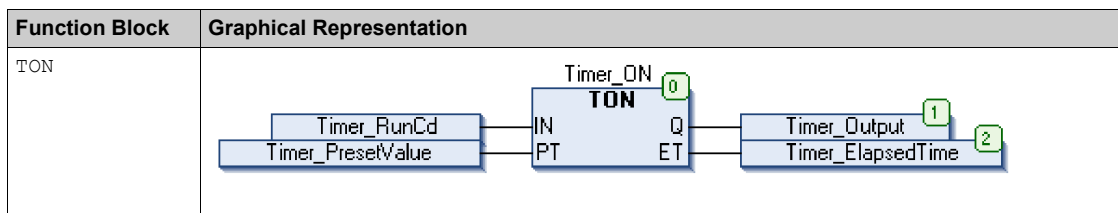
Function	Representation in SoMachine POU ST Editor
SetRTCDrift	<pre>PROGRAM MyProgram_ST VAR myDrift: SINT(-29..29) := 5; myDay: DAY_OF_WEEK := SUNDAY; myHour: HOUR := 12; myMinute: MINUTE; myRTCAdjust: RTCDRIFT_ERROR; END_VAR myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);</pre>

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 <i>SoMachine, Programming Guide</i>).
2	Create the input and output variables and the instance required for the function block: <ul style="list-style-type: none"> • 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:



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

Function Block	Representation in SoMachine POU ST Editor
TON	<pre> 1 PROGRAM MyProgram_ST 2 VAR 3 Timer_ON: TON; // Function Block Instance 4 Timer_RunCd: BOOL; 5 Timer_PresetValue: TIME := T#5S; 6 Timer_Output: BOOL; 7 Timer_ElapsedTime: TIME; 8 END_VAR 1 Timer_ON(2 IN:=Timer_RunCd, 3 PT:=Timer_PresetValue, 4 Q=>Timer_Output, 5 ET=>Timer_ElapsedTime); </pre>



!

%MW

According to the IEC standard, %MW represents a memory word register (for example, a language object of type memory word).

A

analog input

Converts received voltage or current levels into numerical values. You can store and process these values within the logic controller.

application

A program including configuration data, symbols, and documentation.

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`.

ASIC

(application specific integrated circuit) A silicon processor (chip) custom designed especially for an application.

B

BOOL

(boolean) A basic data type in computing. A `BOOL` variable can have one of these values: 0 (`FALSE`), 1 (`TRUE`). A bit that is extracted from a word is of type `BOOL`; for example, `%MW10.4` is a fifth bit of memory word number 10.

Boot application

(boot application) The binary file that contains the application. Usually, it is stored in the controller and allows the controller to boot on the application that the user has generated.

byte

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

C

CFC

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

configuration

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

controller

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

D

digital I/O

(*digital input/output*) An individual circuit connection at the electronic module that corresponds directly to a data table bit. The data table bit holds the value of the signal at the I/O circuit. It gives the control logic digital access to I/O values.

DWORD

(*double word*) Encoded in 32-bit format.

E

element

The short name of the ARRAY element.

expansion bus

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

F

FB

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

firmware

Represents the BIOS, data parameters, and programming instructions that constitute the operating system on a controller. The firmware is stored in non-volatile memory within the controller.

flash memory

A non-volatile memory that can be overwritten. It is stored on a special EEPROM that can be erased and reprogrammed.

function

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

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

function block diagram

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

G**GVL**

(*global variable list*) Manages global variables that can be passed between controllers on an Ethernet TCP/IP Modbus network.

H**hex**

(*hexadecimal*)

HSC

(*high-speed counter*)

I**I/O**

(*input/output*)

ID

(*identifier/identification*)

IEC

(*international electrotechnical commission*) A non-profit and non-governmental international standards organization that prepares and publishes international standards for electrical, electronic, and related technologies.

IEC 61131-3

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

IL

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

INT

(integer) A whole number encoded in 16 bits.

L

LD

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

LED

(light emitting diode) An indicator that illuminates under a low-level electrical charge.

LWORD

(long word) A data type encoded in a 64-bit format.

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.

Modbus

The protocol that allows communications between many devices connected to the same network.

N

network

A system of interconnected devices that share a common data path and protocol for communications.

P**PLC**

(programmable logic controller) An industrial computer used to automate manufacturing, industrial, and other electromechanical processes. PLCs are different from common computers in that they are designed to have multiple input and output arrays and adhere to more robust specifications for shock, vibration, temperature, and electrical interference among other things.

POU

(program organization unit) A variable declaration in source code and a corresponding instruction set. POU's facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POU's 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.

R**RTC**

(real-time clock) A battery-backed time-of-day and calendar clock that operates continuously, even when the controller is not powered for the life of the battery.

run

A command that causes the controller to scan the application program, read the physical inputs, and write to the physical outputs according to solution of the logic of the program.

S**SINT**

(signed integer) A 15-bit value plus sign.

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.

STOP

A command that causes the controller to stop running an application program.

system variable

A variable that provides controller data and diagnostic information and allows sending commands to the controller.

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.

U

UINT

(*unsigned integer*) Encoded in 16 bits.

unlocated variable

A variable that does not have an address (refer to *located variable*).

V

variable

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

W

watchdog

A watchdog is a special timer used to ensure that programs do not overrun their allocated scan time. The watchdog timer is usually set to a higher value than the scan time and reset to 0 at the end of each scan cycle. If the watchdog timer reaches the preset value, for example, because the program is caught in an endless loop, an error is declared and the program stopped.

WORD

A type encoded in a 16-bit format.



B

BOOT_PROJECT_STATUS
Data Types, 62

D

Data Types

BOOT_PROJECT_STATUS, 62
DAY_OF_WEEK, 67
FIRMWARE_VERSION, 61
HOUR, 68
LOCAL_IO_GEN_STATUS, 65
LOCAL_IO_GET_STATUS, 64
MINUTE, 69
PLC_R_APPLICATION_ERROR, 54
PLC_R_BOOT_PROJECT_STATUS, 56
PLC_R_IO_STATUS, 57
PLC_R_STATUS, 53
PLC_R_STOP_CAUSE, 58
PLC_W_COMMAND, 59
RIGHTBUS_GET_STATUS, 66
STOP_WHY, 63
DAY_OF_WEEK
Data Types, 67

F

FIRMWARE_VERSION

Data Types, 61

functions

differences between a function and a function block, 74

Functions

GetBatteryLevel, 27
GetEventsNumber, 28
GetFirmwareVersion, 29
GetHardwareVersion, 30
GetLastStopCause, 31
GetLastStopTime, 32
GetLocalIOStatus, 33
GetPlcFault, 35
GetRightBusStatus, 36
GetSerialNumber, 40
GetShortCutStatus, 41

functions

how to use a function or a function block
in IL language, 75
how to use a function or a function block
in ST language, 79

Functions

InhibitBatLowLed, 47
IsFirstMastColdCycle, 42
IsFirstMastCycle, 43
IsFirstMastWarmCycle, 45
ResetEventsNumber, 48
SetRTCDrift, 49

G

GetBatteryLevel

Functions, 27

GetEventsNumber

Functions, 28

GetFirmwareVersion

Functions, 29

GetHardwareVersion

Functions, 30

GetLastStopCause

Functions, 31

GetLastStopTime

Functions, 32

GetLocalIOStatus

Functions, 33

GetPlcFault
 Functions, 35
GetRightBusStatus
 Functions, 36
GetSerialNumber
 Functions, 40
GetShortCutStatus
 Functions, 41

H

HOUR
 Data Types, 68

I

InhibitBatLowLed
 Functions, 47
IsFirstMastColdCycle
 Functions, 42
IsFirstMastCycle
 Functions, 43
IsFirstMastWarmCycle
 Functions, 45

L

LOCAL_IO_GEN_STATUS
 Data Types, 65
LOCAL_IO_GET_STATUS
 Data Types, 64

M

MINUTE
 Data Types, 69

P

PLC_R
 System Variable, 18
PLC_R_APPLICATION_ERROR
 Data Types, 54
PLC_R_BOOT_PROJECT_STATUS
 Data Types, 56

PLC_R_IO_STATUS
 Data Types, 57
PLC_R_STATUS
 Data Types, 53
PLC_R_STOP_CAUSE
 Data Types, 58
PLC_W
 System Variable, 20
PLC_W_COMMAND
 Data Types, 59

R

ResetEventsNumber
 Functions, 48
RIGHTBUS_GET_STATUS
 Data Types, 66

S

SERIAL_R
 System Variable, 22
SERIAL_W
 System Variable, 23
SetRTCDrift
 Functions, 49
STOP_WHY
 Data Types, 63
System Variable
 PLC_R, 18
 PLC_W, 20
 SERIAL_R, 22
 SERIAL_W, 23
System Variables
 Definition, 13
 Using, 15