

# Modicon M251 Logic Controller System Functions and Variables PLCSystem Library Guide

04/2017



EIO0000001468.01

[www.schneider-electric.com](http://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.

© 2017 Schneider Electric. All Rights Reserved.

---

# Table of Contents

---



	<b>Safety Information</b> .....	<b>7</b>
	<b>About the Book</b> .....	<b>9</b>
<b>Chapter 1</b>	<b>M251 System Variables</b> .....	<b>11</b>
1.1	System Variables: Definition and Use .....	<b>12</b>
	Understanding System Variables .....	<b>13</b>
	Using System Variables .....	<b>15</b>
1.2	PLC_R and PLC_W Structures .....	<b>17</b>
	PLC_R: Controller Read-Only System Variables .....	<b>18</b>
	PLC_W: Controller Read/Write System Variables .....	<b>22</b>
1.3	SERIAL_R and SERIAL_W Structures .....	<b>23</b>
	SERIAL_R[0...1]: Serial Line Read-Only System Variables .....	<b>24</b>
	SERIAL_W[0...1]: Serial Line Read/Write System Variables .....	<b>25</b>
1.4	ETH_R and ETH_W Structures .....	<b>26</b>
	ETH_R: Ethernet Port Read-Only System Variables .....	<b>27</b>
	ETH_W: Ethernet Port Read/Write System Variables .....	<b>33</b>
1.5	TM3_MODULE_R Structure .....	<b>34</b>
	TM3_MODULE_R[0...13]: TM3 Modules Read-Only System Variables .....	<b>34</b>
1.6	TM3_BUS_W Structure .....	<b>35</b>
	TM3_BUS_W: TM3 Bus System Variables .....	<b>35</b>
1.7	PROFIBUS_R Structure .....	<b>36</b>
	PROFIBUS_R: PROFIBUS Read-Only System Variables .....	<b>36</b>
<b>Chapter 2</b>	<b>M251 System Functions</b> .....	<b>37</b>
2.1	M251 Read Functions .....	<b>38</b>
	GetRtc: Get Real Time Clock .....	<b>39</b>
	IsFirstMastColdCycle: Indicate if this Cycle is the First MAST Cold Start Cycle .....	<b>40</b>
	IsFirstMastCycle: Indicate if this Cycle is the First MAST Cycle .....	<b>41</b>
	IsFirstMastWarmCycle: Indicate if this Cycle is the First MAST Warm Start Cycle .....	<b>43</b>
2.2	M251 Write Functions .....	<b>44</b>
	SetRTCDrift: Set Compensation Value to the RTC .....	<b>44</b>
2.3	M251 User Functions .....	<b>46</b>
	DataFileCopy: Copy File Commands .....	<b>47</b>
	ExecuteScript: Run Script Commands .....	<b>50</b>

2.4	TM3 Read Functions .....	52
	TM3_GetModuleBusStatus: Get TM3 Module Bus Status .....	53
	TM3_GetModuleFWVersion: Get TM3 Module Firmware Version ..	54
	TM3_GetModuleInternalStatus: Get TM3 Module Internal Status	55
<b>Chapter 3</b>	<b>M251 PLCSystem Library Data Types .....</b>	<b>57</b>
3.1	PLC_RW System Variables Data Types .....	58
	PLC_R_APPLICATION_ERROR: Detected Application Error Status	
	Codes .....	59
	PLC_R_BOOT_PROJECT_STATUS: Boot Project Status Codes .....	61
	PLC_R_IO_STATUS: I/O Status Codes .....	62
	PLC_R_SDCARD_STATUS: SD Card Slot Status Codes .....	63
	PLC_R_STATUS: Controller Status Codes .....	64
	PLC_R_STOP_CAUSE: From RUN State to Other State Transition	
	Cause Codes .....	65
	PLC_R_TERMINAL_PORT_STATUS: Programming Port Connection	
	Status Codes .....	67
	PLC_R_TM3_BUS_STATE: TM3 Bus Status Codes .....	68
	PLC_W_COMMAND: Control Command Codes .....	69
3.2	DataFileCopy System Variables Data Types .....	70
	DataFileCopyError: Detected Error Codes .....	71
	DataFileCopyLocation: Location Codes .....	72
3.3	ExecScript System Variables Data Types .....	73
	ExecuteScriptError: Detected Error Codes .....	73
3.4	ETH_RW System Variables Data Types .....	74
	ETH_R_FRAME_PROTOCOL: Frame Transmission Protocol Codes ..	75
	ETH_R_IP_MODE: IP Address Source Codes .....	76
	ETH_R_PORT_DUPLEX_STATUS: Transmission Mode Codes .....	77
	ETH_R_PORT_IP_STATUS: Ethernet TCP/IP Port Status Codes ..	78
	ETH_R_PORT_LINK_STATUS: Communication Link Status Codes ..	79
	ETH_R_PORT_SPEED: Communication Speed of the Ethernet Port	
	Codes .....	80
	ETH_R_RUN_IDLE: Ethernet/IP Run and Idle States Codes .....	81
3.5	TM3_MODULE_RW System Variables Data Types .....	82
	TM3_ERR_CODE: TM3 Expansion Module Detected Error Codes ..	83
	TM3_MODULE_R_ARRAY_TYPE: TM3 Expansion Module Read	
	Array Type .....	84
	TM3_MODULE_STATE: TM3 Expansion Module State Codes .....	85
	TM3_BUS_W_IOBUSERRMOD: TM3 bus error mode .....	86

---

3.6	System Function Data Types .....	87
	RTCSETDRIFT_ERROR: SetRTCDrift Function Detected Error Codes .....	87
<b>Appendices</b>	.....	<b>89</b>
<b>Appendix A</b>	<b>Function and Function Block Representation</b> .....	<b>91</b>
	Differences Between a Function and a Function Block .....	92
	How to Use a Function or a Function Block in IL Language .....	93
	How to Use a Function or a Function Block in ST Language .....	96
<b>Glossary</b>	.....	<b>99</b>
<b>Index</b>	.....	<b>107</b>



---

# 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 acquaints you with the system functions and variables offered within the Modicon M251 Logic Controller. The M251 PLCSystem library contains functions and variables to get information from and send commands to the controller system.

This document describes the data type functions and variables of the M251 PLCSystem library.

The following knowledge is required:

- Basic information on the functionality, structure, and configuration of the M251 Logic Controller
- Programming in the FBD, LD, ST, IL, or CFC language
- System variables (global variables)

### Validity Note

This document has been updated for the release of SoMachine V4.3.

### Related Documents

Title of Documentation	Reference Number
SoMachine Programming Guide	<a href="#">EIO0000000067 (ENG);</a> <a href="#">EIO0000000069 (FRE);</a> <a href="#">EIO0000000068 (GER);</a> <a href="#">EIO0000000071 (SPA);</a> <a href="#">EIO0000000070 (ITA);</a> <a href="#">EIO0000000072 (CHS)</a>
Modicon M251 Logic Controller Hardware Guide	<a href="#">EIO0000001486 (ENG);</a> <a href="#">EIO0000001487 (FRE);</a> <a href="#">EIO0000001488 (GER);</a> <a href="#">EIO0000001465 (SPA);</a> <a href="#">EIO0000001490 (ITA);</a> <a href="#">EIO0000001491 (CHS)</a>
Modicon M251 Logic Controller Programming Guide	<a href="#">EIO0000001462 (ENG);</a> <a href="#">EIO0000001463 (FRE);</a> <a href="#">EIO0000001464 (GER);</a> <a href="#">EIO0000001465 (SPA);</a> <a href="#">EIO0000001466 (ITA);</a> <a href="#">EIO0000001467 (CHS)</a>

---

You can download these technical publications and other technical information from our website at <http://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.<sup>1</sup>
- 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.**

<sup>1</sup> 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.**

---

# Chapter 1

## M251 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 M251 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	23
1.4	ETH_R and ETH_W Structures	26
1.5	TM3_MODULE_R Structure	34
1.6	TM3_BUS_W Structure	35
1.7	PROFIBUS_R Structure	36

# Section 1.1

## System Variables: Definition and Use

---

### Overview

This section defines system variables and how to implement them in the Modicon M251 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 TCP, Modbus serial, and EtherNet/IP requests 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

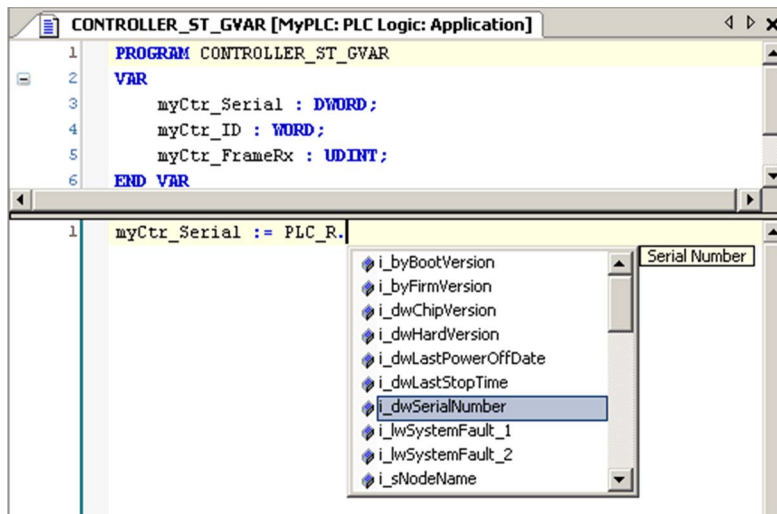
### Introduction

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	22

## 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 <sup>(1)</sup>	Var Name	Type	Comment
60000	i_wVendorID	WORD	Controller Vendor ID. 101A hex = Schneider Electric
60001	i_wProductID	WORD	Controller Reference ID. <b>NOTE:</b> 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	ARRAY[0..3] OF BYTE	Controller Firmware Version [aa.bb.cc.dd]: <ul style="list-style-type: none"> <li>● i_byFirmVersion[0] = aa</li> <li>● ...</li> <li>● i_byFirmVersion[3] = dd</li> </ul>
60006	i_byBootVersion	ARRAY[0..3] OF BYTE	Controller Boot Version [aa.bb.cc.dd]: <ul style="list-style-type: none"> <li>● i_byBootVersion[0] = aa</li> <li>● ...</li> <li>● i_byBootVersion[3] = dd</li> </ul>
60008	i_dwHardVersion	DWORD	Controller Hardware Version.
60010	i_dwChipVersion	DWORD	Controller Coprocessor Version.
60012	i_wStatus	PLC_R_STATUS ( <i>see page 64</i> )	State of the controller.
60013	i_wBootProjectStatus	PLC_R_BOOT_PROJECT_STATUS ( <i>see page 61</i> )	Returns information about the boot application stored in FLASH memory.
60014	i_wLastStopCause	PLC_R_STOP_CAUSE ( <i>see page 65</i> )	Cause of the last transition from RUN to another state.
60015	i_wLastApplicationError	PLC_R_APPLICATION_ERROR ( <i>see page 59</i> )	Cause of the last controller exception.

Modbus Address <sup>(1)</sup>	Var Name	Type	Comment
60016	i_lwSystemFault_1	LWORD	<p>Bit field FFFF FFFF FFFF FFFF hex indicates no error detected.</p> <p>A bit at low level means that an error has been detected:</p> <ul style="list-style-type: none"> <li>● bit 0 = Reserved</li> <li>● bit 1 = TM3 error detected</li> <li>● bit 2 = Ethernet IF1 error detected</li> <li>● bit 3 = Ethernet IF2 error detected</li> <li>● bit 4 = Serial 1 in overcurrent error detected</li> <li>● bit 5 = Reserved</li> <li>● bit 6 = CAN 1 error detected</li> <li>● bit 7 = Reserved</li> <li>● bit 8 = Reserved</li> <li>● bit 9 = TM4 error detected</li> <li>● bit 10 = SD Card error detected</li> <li>● bit 11 = Firewall error detected</li> <li>● bit 12 = DHCP server error detected</li> <li>● bit 13 = OPC UA server error detected</li> </ul>
60020	i_lwSystemFault_2	LWORD	All bits reserved.
60024	i_wIOStatus1	PLC_R_IO_STATUS ( <i>see page 62</i> )	Reserved
60025	i_wIOStatus2	PLC_R_IO_STATUS ( <i>see page 62</i> )	TM3 I/O status.
60026	i_wClockBatterystatus	WORD	<p>Status of the battery of the RTC:</p> <ul style="list-style-type: none"> <li>● 0 = Battery change needed</li> <li>● 100 = Battery fully charged</li> </ul> <p>Other values (1...99) represents the percentage of charge. For example, if the value is 75, it represents that the battery charge is 75%.</p>
60028	i_dwAppliSignature1	DWORD	<p>First DWORD of 4 DWORD signature (16 bytes total).</p> <p>The application signature is generated by the software during build.</p>

Modbus Address <sup>(1)</sup>	Var Name	Type	Comment
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.

<sup>(1)</sup> Not accessible through the application.

n/a	i_sVendorName	STRING (31)	Name of the vendor: "Schneider Electric".
n/a	i_sProductRef	STRING (31)	Reference of the controller.
n/a	i_sNodeName	STRING (99)	Node name on SoMachine Network.
n/a	i_dwLastStopTime	DWORD	The time of the last detected STOP in seconds beginning with January 1, 1970 at 00:00 UTC.
n/a	i_dwLastPowerOffDate	DWORD	The date and time of the last detected power off in seconds beginning with January 1, 1970 at 00:00 UTC.  <b>NOTE:</b> Convert this value into date and time by using the function <code>SysTimeRtcConvertUtcToDate</code> . For more information about Time and Date conversion, refer to the Systeime Library Guide (see <i>SoMachine, Getting &amp; Setting Real Time Clock, SysTimeRtc and SysTimeCore Library Guide</i> ).
n/a	i_uiEventsCounter	UINT	Reserved
n/a	i_wTerminalPortStatus	PLC_R_TERMINAL_PORT_STATUS (see page 67)	Status of the USB Programming Port (USB Mini-B).
n/a	i_wSdCardStatus	PLC_R_SDCARD_STATUS (see page 63)	Status of the SD card.

n/a	i_wUsrFreeFileHdl	WORD	Number of available File Handles. A File Handle is the resource allocated by the system when you open a file.
n/a	i_udiUsrFsTotalBytes	UDINT	User FileSystem total memory size (in bytes). It is the size of the flash memory for the directory "/usr".
n/a	i_udiUsrFsFreeBytes	UDINT	User FileSystem free memory size (in bytes).
n/a	i_uiTM3BusState	PLC_R_TM3_BUS_STATE <i>(see page 68)</i>	TM3 bus state. i_uiTM3BusState can have the following values: <ul style="list-style-type: none"> <li>● 1: TM3_CONF_ERROR Configuration mismatch between physical configuration and SoMachine configuration.</li> <li>● 3: TM3_OK Physical configuration matches SoMachine configuration.</li> <li>● 4: TM3_POWER_SUPPLY_ERROR TM3 bus is not powered (for example when the Logic Controller is powered by USB).</li> </ul>
n/a	i_ExpertIO_RunStop_Input	BYTE	Reserved
n/a	i_x10msClk	BOOL	TimeBase bit of 10 ms. This variable is toggling On/Off with period = 10 ms. The value toggles when the logic controller is in Stop and in Run state.
n/a	i_x100msClk	BOOL	TimeBase bit of 100 ms. This variable is toggling On/Off with period = 100 ms. The value toggles when the logic controller is in Stop and in Run state.
n/a	i_x1sClk	BOOL	TimeBase bit of 1 s. This variable is toggling On/Off with period = 1 s. The value toggles when the logic controller is in Stop and in Run state.

**NOTE:** n/a means that there is no pre-defined Modbus address mapping for this system variable.

## 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_wResetCounterEvent	WORD	Transition from 0 to 1 resets the events counter (PLC_R.i_uiEventsCounter). To reset the counter again, it is necessary to write this register to 0 before another transition from 0 to 1 can take place.
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 <i>(see page 69)</i>	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	24
SERIAL_W[0...1]: Serial Line Read/Write System Variables	25

## 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 M251 Logic Controller:

- Serial\_R[0] refers to serial line
- Serial\_R[1] is reserved

### Variable Structure

This table describes the parameters of the SERIAL\_R[0 . . . 1] system variables:

%MW	Var Name	Type	Comment
<b>Serial Line</b>			
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).
<b>Modbus Specific</b>			
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 M251 Logic Controller:

- Serial\_W[0] refers to serial line
- Serial\_W[1] is reserved

### 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.

# Section 1.4

## ETH\_R and ETH\_W Structures

---

### Overview

This section lists and describes the different system variables included in the `ETH_R` and `ETH_W` structures.

### What Is in This Section?

This section contains the following topics:

Topic	Page
<code>ETH_R</code> : Ethernet Port Read-Only System Variables	27
<code>ETH_W</code> : Ethernet Port Read/Write System Variables	33

## ETH\_R: Ethernet Port Read-Only System Variables

### Variable Structure

This table describes the parameters of the ETH\_R system variable (ETH\_R\_STRUCT type):

%MW	Var Name	Type	Comment
60050	i_byIPAddress	ARRAY[0..3] OF BYTE	IP address of the Ethernet or Ethernet_1 interface [aaa.bbb.ccc.ddd]: <ul style="list-style-type: none"> <li>● i_byIPAddress[0] = aaa</li> <li>● ...</li> <li>● i_byIPAddress[3] = ddd</li> </ul>
60052	i_bySubNetMask	ARRAY[0..3] OF BYTE	Subnet Mask of the Ethernet or Ethernet_1 interface [aaa.bbb.ccc.ddd]: <ul style="list-style-type: none"> <li>● i_bySub-netMask[0] = aaa</li> <li>● ...</li> <li>● i_bySub-netMask[3] = ddd</li> </ul>
60054	i_byGateway	ARRAY[0..3] OF BYTE	Gateway address of the Ethernet or Ethernet_1 interface [aaa.bbb.ccc.ddd]: <ul style="list-style-type: none"> <li>● i_byGateway[0] = aaa</li> <li>● ...</li> <li>● i_byGateway[3] = ddd</li> </ul>
60056	i_byMACAddress	ARRAY[0..5] OF BYTE	MAC address of the Ethernet or Ethernet_1 interface [aa.bb.cc.dd.ee.ff]: <ul style="list-style-type: none"> <li>● i_byMACAddress[0] = aa</li> <li>● ...</li> <li>● i_byMACAddress[5] = ff</li> </ul>
60059	i_sDeviceName	STRING(15)	Name used to get IP address from server.
n/a	i_wIpMode	ETH_R_IP_MODE ( <i>see page 76</i> )	Method used to obtain an IP address.
n/a means that there is no predefined %MW mapping for this system variable.			

%MW	Var Name	Type	Comment
n/a	i_byFDRServerIPAddress	ARRAY[0..3] OF BYTE	The IP address [aaa.bbb.ccc.ddd] of the DHCP or BootP server: <ul style="list-style-type: none"> <li>• i_byFDRServerIPAddress[0] = aaa</li> <li>• ...</li> <li>• i_byFDRServerIPAddress[3] = ddd</li> </ul> Equals 0.0.0.0 if Stored IP or Default IP used.
n/a	i_udiOpenTcpConnections	UDINT	Number of open TCP connections.
n/a	i_udiFramesTransmittedOK	UDINT	Number of frames successfully transmitted. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiFramedReceivedOK	UDINT	Number of frames successfully received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiTransmitBufferErrors	UDINT	Numbers of frames transmitted with detected errors. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiReceiveBufferErrors	UDINT	Numbers of frames received with detected errors. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_wFrameSendingProtocol	ETH_R_FRAME_PROTOCOL (see page 75)	Ethernet protocol configured for frames sending (IEEE 802.3 or Ethernet II).
n/a means that there is no predefined %MW mapping for this system variable.			

%MW	Var Name	Type	Comment
n/a	i_wPortALinkStatus	ETH_R_PORT_LINK_STATUS <i>(see page 79)</i>	Link of the Ethernet Port (0 = No Link, 1 = Link connected to another Ethernet device).
n/a	i_wPortASpeed	ETH_R_PORT_SPEED <i>(see page 80)</i>	Ethernet Port network speed (10Mb/s or 100Mb/s).
n/a	i_wPortADuplexStatus	ETH_R_PORT_DUPLEX_STATUS <i>(see page 77)</i>	Ethernet Port duplex status (0 = Half or 1 = Full duplex).
n/a	i_udiPortACollisions	UDINT	Number of frames involved in one or more collisions and subsequently transmitted successfully. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
<b>Modbus TCP/IP Specific</b>			
n/a	i_udiModbusMessageTransmitted	UDINT	Number of Modbus messages transmitted. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiModbusMessageReceived	UDINT	Number of Modbus messages received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiModbusErrorMessage	UDINT	Modbus detected error messages transmitted and received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
<b>n/a</b> means that there is no predefined %MW mapping for this system variable.			

%MW	Var Name	Type	Comment
<b>EtherNet/IP Specific</b>			
n/a	i_udiETHIP_IOMessagingTransmitted	UDINT	EtherNet/IP Class 1 frames transmitted. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
<b>n/a</b> means that there is no predefined %MW mapping for this system variable. <b>Not used</b> 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.			

%MW	Var Name	Type	Comment
n/a	i_udiETHIP_IOMessagingReceived	UDINT	EtherNet/IP Class 1 frames received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiUCMM_Request	UDINT	EtherNet/IP Unconnected Messages received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiUCMM_Error	UDINT	EtherNet/IP invalid Unconnected Messages received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiClass3_Request	UDINT	EtherNet/IP Class 3 requests received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_udiClass3_Error	UDINT	EtherNet/IP invalid class 3 requests received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_uiAssemblyInstanceInput	UINT	Input Assembly Instance number. See the appropriate Programming Guide of your controller for more information.
n/a	i_uiAssemblyInstanceInputSize	UINT	Input Assembly Instance size. See the appropriate Programming Guide of your controller for more information.
n/a	i_uiAssemblyInstanceOutput	UINT	Output Assembly Instance number. See the appropriate Programming Guide of your controller for more information.

**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.

%MW	Var Name	Type	Comment
n/a	i_uiAssemblyInstanceOutputSize	UINT	Output Assembly Instance size. See the appropriate Programming Guide of your controller for more information.
n/a	i_uiETHIP_ConnectionTimeouts	UINT	Number of connection timeouts. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_ucEipRunIdle	ETH_R_RUN_IDLE (see page 81)	Run (value = 1)/Idle(value = 0) flag for EtherNet/IP class 1 connection.
n/a	i_byMasterIpTimeouts	BYTE	Ethernet Modbus TCP Master timeout events counter. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
n/a	i_byMasterIpLost	BYTE	Ethernet Modbus TCP Master link status: 0 = link OK, 1 = link lost.
n/a	i_wPortAIPStatus	ETH_R_PORT_IP_STATUS (see page 78)	Ethernet TCP/IP port stack status.
n/a	i_byIPAddress_If2	ARRAY[0..3] OF BYTE	IP address of the Ethernet or Ethernet_2 interface [aaa.bbb.ccc.ddd]: <ul style="list-style-type: none"> <li>● i_byIPAddress[0] = aaa</li> <li>● ...</li> <li>● i_byIPAddress[3] = ddd</li> </ul>
n/a	i_bySubNetMask_If2	ARRAY[0..3] OF BYTE	Subnet Mask of the Ethernet or Ethernet_2 interface [aaa.bbb.ccc.ddd]: <ul style="list-style-type: none"> <li>● i_bySub-netMask[0] = aaa</li> <li>● ...</li> <li>● i_bySub-netMask[3] = ddd</li> </ul>

**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.

%MW	Var Name	Type	Comment
n/a	i_byGateway_If2	ARRAY[0..3] OF BYTE	Gateway address of the Ethernet or Ethernet_2 interface [aaa.bbb.ccc.ddd]: <ul style="list-style-type: none"> <li>● i_byGateway[0] = aaa</li> <li>● ...</li> <li>● i_byGateway[3] = ddd</li> </ul>
n/a	i_byMACAddress_If2	ARRAY[0..5] OF BYTE	MAC address of the Ethernet or Ethernet_2 interface [aa.bb.cc.dd.ee.ff]: <ul style="list-style-type: none"> <li>● i_byMACAddress[0] = aa</li> <li>● ...</li> <li>● i_byMACAddress[5] = ff</li> </ul>
n/a	i_sDeviceName_If2	STRING(15)	Name used to get IP address from server.
n/a	i_wIpMode_If2	ETH_R_IP_MODE (see page 76)	Method used to obtain an IP address.
n/a	i_wPortALinkStatus_If2	ETH_R_PORT_LINK_STATUS (see page 79)	Link of the Ethernet Port (0 = No Link, 1 = Link connected to another Ethernet device).
n/a	i_wPortASpeed_If2	ETH_R_PORT_SPEED (see page 80)	Ethernet Port network speed (10Mb/s or 100Mb/s).
n/a	i_wPortADuplexStatus_If2	ETH_R_PORT_DUPLEX_STATUS (see page 77)	Ethernet Port duplex status (0 = Half or 1 = Full duplex).
n/a	i_wPortAIpStatus_If2	ETH_R_PORT_IP_STATUS (see page 78)	Ethernet TCP/IP port stack status.
<p><b>n/a</b> means that there is no predefined %MW mapping for this system variable.  <b>Not used</b> 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.</p>			

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



## ETH\_W: Ethernet Port Read/Write System Variables

### Variable Structure

This table describes the parameters of the ETH\_W system variable (ETH\_W\_STRUCT type):

%MW	Var Name	Type	Comment
n/a	q_wResetCounter	WORD	Transition from 0 to 1 resets all ETH_R counters. To reset 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.

## Section 1.5

### TM3\_MODULE\_R Structure

#### TM3\_MODULE\_R[0...13]: TM3 Modules Read-Only System Variables

##### Introduction

The TM3\_MODULE\_R is an array of 14 TM3\_MODULE\_R\_STRUCT type. Each element of the array returns diagnostic system variables for the corresponding TM3 expansion module.

For the Modicon M251 Logic Controller:

- TM3\_MODULE\_R[0] refers to the TM3 expansion module 0
- ...
- TM3\_MODULE\_R[13] refers to the TM3 expansion module 13

##### Variable Structure

The following table describes the parameters of the TM3\_MODULE\_R[0...13] system variable:

%MW	Var Name	Type	Comment
n/a	i_wProductID	WORD	TM3 expansion module ID.
n/a	i_wModuleState	TM3_MODULE_STATE (see page 85)	Describes the state of the TM3 module.

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

## Section 1.6

### TM3\_BUS\_W Structure

#### TM3\_BUS\_W: TM3 Bus System Variables

##### Variable Structure

This table describes the parameters of the `TM3_BUS_W` system variable (`TM3_BUS_W_STRUCT` type):

Var Name	Type	Comment
<code>q_wIOBusErrPassiv</code>	<code>TM3_BUS_W_IOBUSERRMOD</code>	When set to <code>ERR_ACTIVE</code> (the default), bus errors detected on TM3 expansion modules stop all I/O exchanges. When set to <code>ERR_PASSIVE</code> , passive I/O error handling is used: the controller attempts to continue data bus exchanges.
<code>q_wIOBusRestart</code>	<code>TM3_BUS_W_IOBUSINIT</code>	When set to 1, restarts the I/O expansion bus. This is only necessary when <code>q_wIOBusErrPassiv</code> is set to <code>ERR_ACTIVE</code> and at least one bit of <code>TM3_MODULE_R[i].i_wModuleState</code> is set to <code>TM3_BUS_ERROR</code> .

For more information, refer to I/O Configuration General Description (*see Modicon M251 Logic Controller, Programming Guide*).

## Section 1.7

### PROFIBUS\_R Structure

#### PROFIBUS\_R: PROFIBUS Read-Only System Variables

##### Variable Structure

This table describes the parameters of the PROFIBUS\_R system variable (PROFIBUS\_R\_STRUCT type):

%MW	Var Name	Type	Comment
n/a	i_wPNOIdentifier	WORD	Slave identification code.
n/a	i_wBusAdr	UINT	PROFIBUS slave address.
n/a	i_CommState	UDINT	Value representing the state of the PROFIBUS module: <ul style="list-style-type: none"> <li>● 0x00: Unknown</li> <li>● 0x01: Not configured</li> <li>● 0x02: Stop</li> <li>● 0x03: Idle</li> <li>● 0x04: Operate</li> </ul>
n/a	i_CommError	UDINT	Communication error code.
n/a	i_ErrorCount	UDINT	Communication error counter.

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

---

# Chapter 2

## M251 System Functions

---

### Overview

This chapter describes the system functions included in the M251 PLCSystem library.

### What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
2.1	M251 Read Functions	38
2.2	M251 Write Functions	44
2.3	M251 User Functions	46
2.4	TM3 Read Functions	52

# Section 2.1

## M251 Read Functions

---

### Overview

This section describes the read functions included in the M251 PLCSystem library.

### What Is in This Section?

This section contains the following topics:

Topic	Page
GetRtc: Get Real Time Clock	39
IsFirstMastColdCycle: Indicate if this Cycle is the First MAST Cold Start Cycle	40
IsFirstMastCycle: Indicate if this Cycle is the First MAST Cycle	41
IsFirstMastWarmCycle: Indicate if this Cycle is the First MAST Warm Start Cycle	43

## GetRtc: Get Real Time Clock

### Function Description

This function returns RTC time in seconds in UNIX format (time expired in seconds since January 1, 1970 at 00:00 UTC).

### 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 91).

### I/O Variable Description

The following table describes the I/O variable:

Output	Type	Comment
GetRtc	DINT	RTC in seconds in UNIX format.

### Example

The following example describes how to get the RTC value:

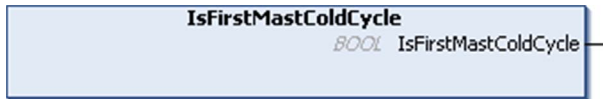
```
VAR
    MyRTC : DINT := 0;
END_VAR
MyRTC := GetRtc();
```

## IsFirstMastColdCycle: Indicate if this 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 91).

### 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 41).



## IsFirstMastCycle: Indicate if this 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 91)*.

### 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*)

```

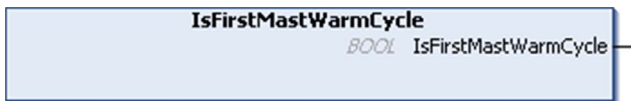
```
END_IF;
IF (MyIsFirstMastColdCycle) THEN
(*This is the first Mast Cycle after a Cold Start: all variables are set
to their initialization values including the Retain Variables*)
(*=> initialize the needed variables so that your application runs as
expected in this case*)
END_IF;
IF (MyIsFirstMastCycle) 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: Indicate if this 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 91).

### 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 41).

## Section 2.2

### M251 Write Functions

#### SetRTCDrift: Set Compensation Value to the RTC

##### Function Description

This function accelerates or slows down the frequency of the RTC to give control to the application for RTC compensation, depending on the operating environment (temperature, ...). The compensation value is given in seconds per week. It can be positive (accelerate) or negative (slow down).

**NOTE:** The `SetRTCDrift` function must be called only once. Each new call replaces the compensation value by the new one. The value is kept in the logic controller hardware while the RTC is powered by the main supply or by the battery. If both battery and power supply are removed, the RTC compensation value is not available.

##### 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 91](#)).

##### I/O Variables Description

This table describes the input parameters:

Inputs	Type	Comment
RtcDrift	SINT (-36...+73)	Correction in seconds per week (-36 ... +73).

**NOTE:** The parameters `Day`, `Hour`, and `Minute` are used only to ensure backwards compatibility.

**NOTE:** If the value entered for `RtcDrift` exceeds the limit value, the logic controller firmware sets the value to its maximum value.

This table describes the output variable:

Output	Type	Comment
SetRTCDrift	RTCSETDRIFT_ERROR ( <i>see page 87</i> )	Returns RTC_OK (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. It accelerates the RTC by 4 seconds a week (18 seconds a month).

```

VAR
    MyRTCDrift : SINT (-36...+73) := 0;
    MyDay : DAY_OF_WEEK;
    MyHour : HOUR;
    MyMinute : MINUTE;
END_VAR

IF IsFirstMastCycle() THEN
    MyRTCDrift := 4;
    MyDay := 0;
    MyHour := 0;
    MyMinute := 0;
    SetRTCDrift(MyRTCDrift, MyDay, MyHour, MyMinute);
END_IF

```

## Section 2.3

### M251 User Functions

---

#### Overview

This section describes the `DataFileCopy` and `ExecuteScript` functions included in the M251 PLCSystem library.

#### What Is in This Section?

This section contains the following topics:

Topic	Page
<code>DataFileCopy</code> : Copy File Commands	47
<code>ExecuteScript</code> : Run Script Commands	50

## DataFileCopy: Copy File Commands

### Function Block Description

This function block copies memory data to a file and vice versa. The file is located either within the internal file system or an external file system (SD card).

The DataFileCopy function block can:

- Read data from a formatted file or
- Copy data from memory buffer to a formatted file. For further information, refer to Flash Memory Organization (see *Modicon M251 Logic Controller, Programming 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 91](#)).

### I/O Variable Description

This table describes the input variables:

Input	Type	Comment
xExecute	BOOL	On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates.
sFileName	STRING	File name without extension (the extension <i>.DTA</i> is automatically added). Use only a...z, A...Z, 0...9 alphanumeric characters.
xRead	BOOL	TRUE: copy from file to memory. FALSE: copy from memory to file.
xSecure	BOOL	TRUE: The MAC address is always stored in the file. Only a controller with the same MAC address can read from the file. FALSE: Another controller with the same type of memory can read from the file.
iLocation	INT	0: the file location is <i>/usr/DTA</i> in internal file system. 1: the file location is <i>/usr/DTA</i> in external file system (SD card).

Input	Type	Comment
uiSize	UINT	Indicates the size in bytes. Maximum is 65534 bytes. Only use addresses of variables conforming to IEC 6113-1 (variables, arrays, structures), for example: Variable : int; uiSize := SIZEOF (Variable);
dwAdd	DWORD	Indicates the address in the memory. Only use addresses of variables conforming to IEC 6113-1 (variables, arrays, structures), for example: Variable : int; dwAdd := ADR (Variable);

## WARNING

### UNINTENDED EQUIPMENT OPERATION

Verify that the memory location is of the correct size and the file is of the correct type before copying the file to memory.

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

This table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE = indicates that the action is successfully completed.
xBusy	BOOL	TRUE = indicates that the function block is running.
xError	BOOL	TRUE = indicates that an error is detected and the function block aborted the action.
eError	DataFileCopyError ( <i>see page 71</i> )	Indicates the type of the data file copy detected error.

**NOTE:** If you write to memory variable within the area of the file write, you generate a CRC detected error.



**Example**

This example describes how to copy file commands:

```
VAR
LocalArray : ARRAY [0..29] OF BYTE;
myFileName: STRING := 'exportfile';
EXEC_FLAG: BOOL;
DataFileCopy: DataFileCopy;
END_VAR
DataFileCopy(
xExecute:= EXEC_FLAG,
sFileName:= myFileName,
xRead:= FALSE,
xSecure:= FALSE,
iLocation:= DFCL_INTERNAL,
uiSize:= SIZEOF(LocalArray),
dwAdd:= ADR(LocalArray),
xDone=> ,
xBusy=> ,
xError=> ,
eError=> );
```

## ExecuteScript: Run Script Commands

### Function Block Description

This function block can run the following SD card script commands:

- Download
- Upload
- SetNodeName
- Delete
- Reboot
- ChangeModbusPort

For information on the required script file format, refer to Script Files for SD Cards.

### 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 91)*.

### I/O Variable Description

This table describes the input variables:

Input	Type	Comment
xExecute	BOOL	On detection of a rising edge, starts the function block execution. On detection of a falling edge, resets the outputs of the function block when its execution terminates.
sCmd	STRING	SD card script command syntax. Simultaneous command executions are not allowed: if a command is being executed from another function block or from an SD card script then the function block queues the command and does not execute it immediately. <b>NOTE:</b> An SD card script executed from an SD card is considered as being executed until the SD card has been removed.

This table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE indicates that the action is successfully completed.
xBusy	BOOL	TRUE indicates that the function block is running.
xError	BOOL	TRUE indicates error detection; the function block aborts the action.
eError	ExecuteScriptError (see page 73)	Indicates the type of the execute script detected error.

### Example

This example describes how to execute an Upload script command:

```

VAR
EXEC_FLAG: BOOL;
ExecuteScript: ExecuteScript;
END_VAR
ExecuteScript(
xExecute:= EXEC_FLAG,
sCmd:= 'Upload "/usr/Syslog/*"',
xDone=> ,
xBusy=> ,
xError=> ,
eError=> );

```

## Section 2.4

### TM3 Read Functions

---

#### Overview

This section describes the TM3 read functions included in the M251 PLCSystem library.

#### What Is in This Section?

This section contains the following topics:

Topic	Page
TM3_GetModuleBusStatus: Get TM3 Module Bus Status	53
TM3_GetModuleFWVersion: Get TM3 Module Firmware Version	54
TM3_GetModuleInternalStatus: Get TM3 Module Internal Status	55

## TM3\_GetModuleBusStatus: Get TM3 Module Bus Status

### Function Description

This function returns the bus status of the module. The index of the module is given as an input parameter.

### 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 91](#)).

### I/O Variable Description

The following table describes the input variable:

Input	Type	Comment
ModuleIndex	BYTE	Index of the module (0 for the first expansion, 1 for the second, and so on).

The following table describes the output variable:

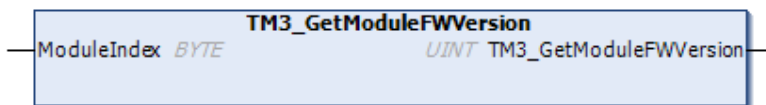
Output	Type	Comment
TM3_GetModuleBusStatus	TM3_ERR_CODE ( <a href="#">see page 83</a> )	Returns TM3_OK (00 hex) if command is correct otherwise returns the ID code of the detected error.

## TM3\_GetModuleFWVersion: Get TM3 Module Firmware Version

### Function Description

This function returns the firmware version of a specified TM3 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 91).

### I/O Variable Description

The following table describes the input variables:

Input	Type	Comment
ModuleIndex	BYTE	Index of the module (0 for the first expansion, 1 for the second, and so on).

The following table describes the output variable:

Output	Type	Comment
TM3_GetModuleFWVersion	UINT	Returns the firmware version of the module, or <code>FFFF hex</code> if the information cannot be read. For example, <code>001A hex</code> indicates firmware version 26.

## TM3\_GetModuleInternalStatus: Get TM3 Module Internal Status

### Function Description

This function fills the `pStatusBuffer` with the status table of the module `ModuleIndex`.

### 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 91)*.

### I/O Variable Description

<b>⚠ WARNING</b>
<b>UNINTENDED EQUIPMENT OPERATION</b>
Ensure that the <code>pStatusBuffer</code> is allocated.
<b>Failure to follow these instructions can result in death, serious injury, or equipment damage.</b>

The following table describes the input variables:

Input	Type	Comment
<code>ModuleIndex</code>	BYTE	Index of the module (0 for the first expansion, 1 for the second, and so on).
<code>StatusOffset</code>	BYTE	Offset of the first status to be read in the status table.
<code>StatusSize</code>	BYTE	Number of bytes to be read in the status table.
<code>pStatusBuffer</code>	POINTER TO BYTE	Buffer containing the read status table.

The following table describes the output variable:

Output	Type	Comment
<code>TM3_GetModuleInternalStatus</code>	TM3_ERR_CODE (see page 83)	Returns <code>TM3_OK</code> (00 hex) if command is correct otherwise returns the ID code of the error.

### Example

The following example describes how to get the module internal status:

```
VAR  
AMM3HT_Channel1_Input_Status: BYTE;  
END_VAR  
TM3_GetModuleInternalStatus(0, 1, 1, ADR(AMM3HT_Channel1_Input_Status));
```



---

# Chapter 3

## M251 PLCSystem Library Data Types

---

### Overview

This chapter describes the data types of the M251 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 M251 PLCSystem Library (PLC\_R, PLC\_W,...).
- System function data types are used by the read/write system functions (*see page 37*) of the M251 PLCSystem Library.

### What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
3.1	PLC_RW System Variables Data Types	58
3.2	DataFileCopy System Variables Data Types	70
3.3	ExecScript System Variables Data Types	73
3.4	ETH_RW System Variables Data Types	74
3.5	TM3_MODULE_RW System Variables Data Types	82
3.6	System Function Data Types	87

# Section 3.1

## PLC\_RW 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_APPLICATION_ERROR: Detected Application Error Status Codes	59
PLC_R_BOOT_PROJECT_STATUS: Boot Project Status Codes	61
PLC_R_IO_STATUS: I/O Status Codes	62
PLC_R_SDCARD_STATUS: SD Card Slot Status Codes	63
PLC_R_STATUS: Controller Status Codes	64
PLC_R_STOP_CAUSE: From RUN State to Other State Transition Cause Codes	65
PLC_R_TERMINAL_PORT_STATUS: Programming Port Connection Status Codes	67
PLC_R_TM3_BUS_STATE: TM3 Bus Status Codes	68
PLC_W_COMMAND: Control Command Codes	69

## 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: <b>1. Build → Clean All</b> <b>2. Export/Import your application.</b> <b>3. Upgrade SoMachine to the latest version.</b>
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: <b>1. Build → Clean All</b> <b>2. Export/Import your application.</b> <b>3. Upgrade SoMachine to the latest version.</b>
PLC_R_APP_ERR_ILLEGAL_INSTRUCTION	0050 hex	Undefined instruction detected.	Debug your application to resolve the problem.
PLC_R_APP_ERR_ACCESS_VIOLATION	0051 hex	Attempted access to reserved memory area.	Debug your application to resolve the problem.

Enumerator	Value	Comment	What to do
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_APP_ERR_EXPIO_EVENTS_COUNT_EXCEEDED	4E20 hex	Too many events on expert I/Os are detected.	Reduce the number of event tasks.
PLC_R_APP_ERR_APPLICATION_VERSION_MISMATCH	4E21 hex	Mismatch in the application version detected.	The application version in the logic controller does not match the version in SoMachine. Refer to Applications (see <i>SoMachine, Programming Guide</i> ).

## 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	Boot project does not exist in Flash memory.
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.
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.
PLC_R_IO_POWER_SUPPLY_FAULT	0004 hex	Inputs/Outputs power supply error detected.

## PLC\_R\_SDCARD\_STATUS: SD Card Slot Status Codes

### Enumerated Type Description

The PLC\_R\_SDCARD\_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
NO_SDCARD	0000 hex	No SD card detected in the slot or the slot is not connected.
SDCARD_READONLY	0001 hex	SD card is in read-only mode.
SDCARD_READWRITE	0002 hex	SD card is in read/write mode.
SDCARD_ERROR	0003 hex	Error detected in the SD card.

## 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\_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.PLC_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_FROM_NETWORK_REQUEST	0A hex	Stopped after a request from the network, the controller Web server, or PLC_W command.	–
PLC_R_STOP_FROM_INPUT	0B hex	Stop required by a controller input.	–
PLC_R_STOP_FROM_RUN_STOP_SWITCH	0C hex	Stop required by the controller switch.	
PLC_R_STOP_REASON_RETAIN_MISMATCH	0D 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.

---

Enumerator	Value	Comment	What to do
PLC_R_STOP_REASON_ BOOT_APPLI_MISMATCH	0E 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	0F hex	Stopped after a power interruption.	–

For more information on the reasons why the controller has stopped, refer to the Controller State Description (*see Modicon M251 Logic Controller, Programming Guide*).

## PLC\_R\_TERMINAL\_PORT\_STATUS: Programming Port Connection Status Codes

### Enumerated Type Description

The PLC\_R\_TERMINAL\_PORT\_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
TERMINAL_NOT_CONNECTED	00 hex	No PC is connected to the programming port.
TERMINAL_CONNECTION_IN_PROGRESS	01 hex	Connection is in progress.
TERMINAL_CONNECTED	02 hex	PC is connected to the programming port.
TERMINAL_ERROR	0F hex	Error detected during connection.

## PLC\_R\_TM3\_BUS\_STATE: TM3 Bus Status Codes

### Enumerated Type Description

The PLC\_R\_TM3\_BUS\_STATE enumeration data type contains the following values:

Enumerator	Value	Comment
TM3_CONF_ERROR	01 hex	Error detected due to mismatch in the physical configuration and the configuration in SoMachine.
TM3_OK	03 hex	The physical configuration and the configuration in SoMachine match.
TM3_POWER_SUPPLY_ERROR	04 hex	Error detected in power supply.

## 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

### DataFileCopy System Variables Data Types

---

#### Overview

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

#### What Is in This Section?

This section contains the following topics:

Topic	Page
DataFileCopyError: Detected Error Codes	71
DataFileCopyLocation: Location Codes	72

## DataFileCopyError: Detected Error Codes

### Enumerated Type Description

The `DataFileCopyError` enumeration data type contains the following values:

Enumerator	Value	Description
<code>ERR_NO_ERR</code>	00 hex	No error detected.
<code>ERR_FILE_NOT_FOUND</code>	01 hex	The file does not exist.
<code>ERR_FILE_ACCESS_REFUSED</code>	02 hex	The file cannot be opened.
<code>ERR_INCORRECT_SIZE</code>	03 hex	The request size is not the same as size read from file.
<code>ERR_CRC_ERR</code>	04 hex	The CRC is not correct and the file is assumed to be corrupted.
<code>ERR_INCORRECT_MAC</code>	05 hex	The controller attempting to read from the file does not have the same MAC address as that contained in the file.

## DataFileCopyLocation: Location Codes

### Enumerated Type Description

The `DataFileCopyLocation` enumeration data type contains the following values:

Enumerator	Value	Description
<code>DFCL_INTERNAL</code>	00 hex	Data file with DTA extension is located in <i>/usr/Dta</i> directory.
<code>DFCL_EXTERNAL</code>	01 hex	Data file with DTA extension is located in <i>/sd0/usr/Dta</i> directory.
<code>DFCL_TBD</code>	02 hex	Not used.



## Section 3.3

### ExecScript System Variables Data Types

#### ExecuteScriptError: Detected Error Codes

##### Enumerated Type Description

The `ExecuteScriptError` enumeration data type contains the following values:

Enumerator	Value	Description
<code>CMD_OK</code>	00 hex	No error detected.
<code>ERR_CMD_UNKNOWN</code>	01 hex	The command is not recognized.
<code>ERR_SD_CARD_MISSING</code>	02 hex	SD card is not present.
<code>ERR_SEE_FWLOG</code>	03 hex	There was an error detected during command execution, see <code>FwLog.txt</code> . For more information, refer to File Type ( <i>see Modicon M251 Logic Controller, Programming Guide</i> ).
<code>ERR_ONLY_ONE_COMMAND_ALLOWED</code>	04 hex	An attempt was made to execute several scripts simultaneously.
<code>CMD_BEING_EXECUTED</code>	05 hex	A script is already in progress.

## Section 3.4

### ETH\_RW System Variables Data Types

---

#### Overview

This section lists and describes the system variable data types included in the `ETH_R` and `ETH_W` structures.

#### What Is in This Section?

This section contains the following topics:

Topic	Page
ETH_R_FRAME_PROTOCOL: Frame Transmission Protocol Codes	75
ETH_R_IP_MODE: IP Address Source Codes	76
ETH_R_PORT_DUPLEX_STATUS: Transmission Mode Codes	77
ETH_R_PORT_IP_STATUS: Ethernet TCP/IP Port Status Codes	78
ETH_R_PORT_LINK_STATUS: Communication Link Status Codes	79
ETH_R_PORT_SPEED: Communication Speed of the Ethernet Port Codes	80
ETH_R_RUN_IDLE: Ethernet/IP Run and Idle States Codes	81

## ETH\_R\_FRAME\_PROTOCOL: Frame Transmission Protocol Codes

### Enumerated Type Description

The `ETH_R_FRAME_PROTOCOL` enumeration data type contains the following values:

Enumerator	Value	Comment
<code>ETH_R_802_3</code>	00 hex	The protocol used for frame transmission is IEEE 802.3.
<code>ETH_R_ETHERNET_II</code>	01 hex	The protocol used for frame transmission is Ethernet II.

## ETH\_R\_IP\_MODE: IP Address Source Codes

### Enumerated Type Description

The ETH\_R\_IP\_MODE enumeration data type contains the following values:

Enumerator	Value	Comment
ETH_R_STORED	00 hex	Stored IP address is used.
ETH_R_BOOTP	01 hex	Bootstrap protocol is used to get an IP address.
ETH_R_DHCP	02 hex	DHCP protocol is used to get an IP address.
ETH_DEFAULT_IP	FF hex	Default IP address is used.

## ETH\_R\_PORT\_DUPLEX\_STATUS: Transmission Mode Codes

### Enumerated Type Description

The ETH\_R\_PORT\_DUPLEX\_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
ETH_R_PORT_HALF_DUPLEX	00 hex	Half duplex transmission mode is used.
ETH_R_FULL_DUPLEX	01 hex	Full duplex transmission mode is used.
ETH_R_PORT_NA_DUPLEX	03 hex	No duplex transmission mode is used.

## ETH\_R\_PORT\_IP\_STATUS: Ethernet TCP/IP Port Status Codes

### Enumerated Type Description

The ETH\_R\_PORT\_IP\_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
WAIT_FOR_PARAMS	00 hex	Waiting for parameters.
WAIT_FOR_CONF	01 hex	Waiting for configuration.
DATA_EXCHANGE	02 hex	Ready for data exchange.
ETH_ERROR	03 hex	Ethernet TCP/IP port error detected (cable disconnected, invalid configuration, and so on).
DUPLICATE_IP	04 hex	IP address already used by another equipment.

## ETH\_R\_PORT\_LINK\_STATUS: Communication Link Status Codes

### Enumerated Type Description

The ETH\_R\_PORT\_LINK\_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
ETH_R_LINK_DOWN	00 hex	Communication link not available to another device.
ETH_R_LINK_UP	01 hex	Communication link available to another device.

## ETH\_R\_PORT\_SPEED: Communication Speed of the Ethernet Port Codes

### Enumerated Type Description

The ETH\_R\_PORT\_SPEED enumeration data type contains the following values:

Enumerator	Value	Comment
ETH_R_SPEED_NA	0 dec	Network speed is 0 megabits per second.
ETH_R_SPEED_10_MB	10 dec	Network speed is 10 megabits per second.
ETH_R_100_MB	100 dec	Network speed is 100 megabits per second.



## ETH\_R\_RUN\_IDLE: Ethernet/IP Run and Idle States Codes

### Enumerated Type Description

The `ETH_R_RUN_IDLE` enumeration data type contains the following values:

Enumerator	Value	Comment
IDLE	00 hex	EtherNet/IP connection is idle.
RUN	01 hex	EtherNet/IP connection is running.

## Section 3.5

### TM3\_MODULE\_RW System Variables Data Types

---

#### Overview

This section lists and describes the system variable data types included in the `TM3_MODULE_R` and `TM3_MODULE_W` structures.

#### What Is in This Section?

This section contains the following topics:

Topic	Page
TM3_ERR_CODE: TM3 Expansion Module Detected Error Codes	83
TM3_MODULE_R_ARRAY_TYPE: TM3 Expansion Module Read Array Type	84
TM3_MODULE_STATE: TM3 Expansion Module State Codes	85
TM3_BUS_W_IOBUSERRMOD: TM3 bus error mode	86

## TM3\_ERR\_CODE: TM3 Expansion Module Detected Error Codes

### Enumerated Type Description

The `TM3_ERR_CODE` enumeration data type contains the following values:

Enumerator	Value	Comment
<code>TM3_NO_ERR</code>	00 hex	Last bus exchange with the expansion module was successful.
<code>TM3_ERR_FAILED</code>	01 hex	Error detected due to the last bus exchange with the expansion module was unsuccessful.
<code>TM3_ERR_PARAMETER</code>	02 hex	Parameter error detected in the last bus exchange with the module.
<code>TM3_ERR_COK</code>	03 hex	Temporary or permanent hardware error detected on one of the TM3 expansion modules.
<code>TM3_ERR_BUS</code>	04 hex	Bus error detected in the last bus exchange with the expansion module.

## TM3\_MODULE\_R\_ARRAY\_TYPE: TM3 Expansion Module Read Array Type

### Description

The `TM3_MODULE_R_ARRAY_TYPE` is an array of 0...13 `TM3_MODULE_R_STRUCT`.

## TM3\_MODULE\_STATE: TM3 Expansion Module State Codes

### Enumerated Type Description

The TM3\_MODULE\_STATE enumeration data type contains the following values:

Enumerator	Value	Comment
TM3_EMPTY	00 hex	No module.
TM3_CONF_ERROR	01 hex	Physical expansion module does not match with the one configured in SoMachine.
TM3_BUS_ERROR	02 hex	Bus error detected in the last exchange with the module.
TM3_OK	03 hex	Last bus exchange with this module was successful.
TM3_MISSING_OPT_MOD	05 hex	Optional module is not physically present.

## TM3\_BUS\_W\_IOBUSERRMOD: TM3 bus error mode

### Enumerated Type Description

The TM3\_BUS\_W\_IOBUSERRMOD enumeration data type contains the following values:

Enumerator	Value	Comment
IOBUS_ERR_ACTIVE	00 hex	Active mode. The logic controller stops all I/O exchanges on the TM3 bus on detection of a permanent error. Refer to I/O Configuration General Description ( <i>see Modicon M251 Logic Controller, Programming Guide</i> ).
IOBUS_ERR_PASSIVE	01 hex	Passive mode. I/O exchanges continue on the TM3 bus even if an error is detected.

## Section 3.6

### System Function Data Types

#### RTCSETDRIFT\_ERROR: setRTCDrift Function Detected Error Codes

##### Enumerated Type Description

The RTCSETDRIFT\_ERROR enumeration data type contains the following values:

Enumerator	Value	Comment
RTC_OK	00 hex	RTC drift correctly configured.
RTC_BAD_DAY	01 hex	Not used.
RTC_BAD_HOUR	02 hex	Not used.
RTC_BAD_MINUTE	03 hex	Not used.
RTC_BAD_DRIFT	04 hex	RTC Drift parameter out of range.
RTC_INTERNAL_ERROR	05 hex	RTC Drift settings rejected on internal error detected.





---

# 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	92
How to Use a Function or a Function Block in IL Language	93
How to Use a Function or a Function Block in ST Language	96

## 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. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POU's ( <i>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"> <li>type the name of the function in the operator column (left field), or</li> <li>use the <b>Input Assistant</b> to select the function (select <b>Insert Box</b> in the context menu).</li> </ul>
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 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/> <pre> 1  IsFirstMast Cycle    ST          FirstCycle </pre>
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/> <pre> 1  LD          myDrift    SetRTCDrift myDay            myHour            myMinute    ST          myDiag </pre>

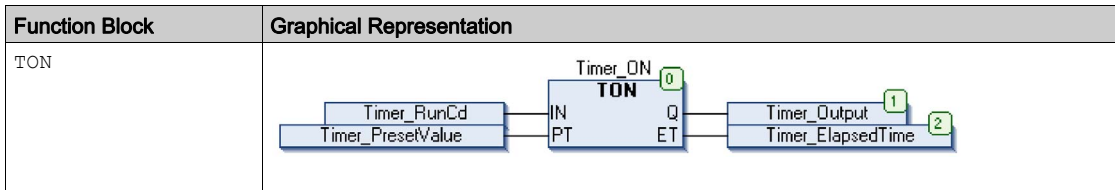
### Using a Function Block in IL Language

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

Step	Action
1	Open or create a new POU in Instruction List language. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs ( <i>see SoMachine, Programming Guide</i> ).
2	Create the variables that the function block requires, including the instance name.

Step	Action
3	<p>Function Blocks are called using a CAL instruction:</p> <ul style="list-style-type: none"> <li>● Use the <b>Input Assistant</b> to select the FB (right-click and select <b>Insert Box</b> in the context menu).</li> <li>● Automatically, the CAL instruction and the necessary I/O are created.</li> </ul> <p>Each parameter (I/O) is an instruction:</p> <ul style="list-style-type: none"> <li>● Values to inputs are set by ":=".</li> <li>● Values to outputs are set by "=&gt;".</li> </ul>
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:



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

Function Block	Representation in 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 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 </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.

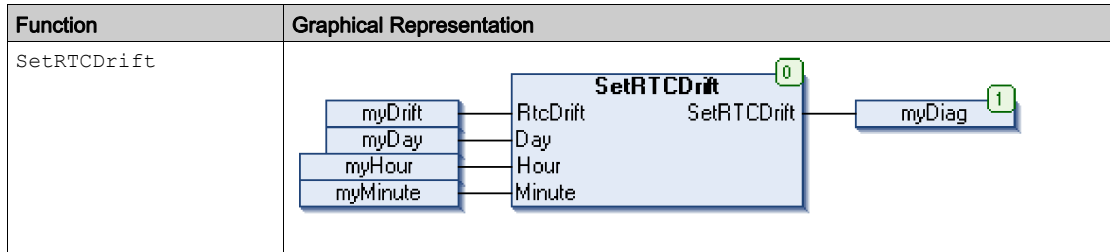
Function `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

### 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. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POU's ( <i>see SoMachine, Programming Guide</i> ).
2	Create the variables that the function requires.
3	Use the general syntax in the <b>POU ST Editor</b> 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 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>

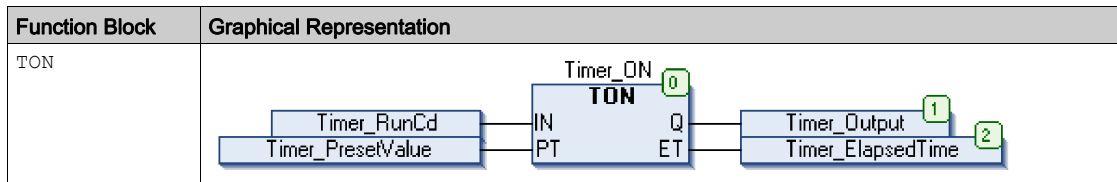


## Using a Function Block in ST Language

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. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation ( <i>see 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"> <li>• Input variables are the input parameters required by the function block</li> <li>• Output variables receive the value returned by the function block</li> </ul>
3	Use the general syntax in the <b>POU ST Editor</b> 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 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=&gt;Timer_Output, 5      ET=&gt;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

### **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`.

## 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.

### **BOOTP**

(*bootstrap protocol*) A UDP network protocol that can be used by a network client to automatically obtain an IP address (and possibly other data) from a server. The client identifies itself to the server using the client MAC address. The server, which maintains a pre-configured table of client device MAC addresses and associated IP addresses, sends the client its pre-configured IP address. BOOTP was originally used as a method that enabled diskless hosts to be remotely booted over a network. The BOOTP process assigns an infinite lease of an IP address. The BOOTP service utilizes UDP ports 67 and 68.

### **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.

### 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.

### controller

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

### CRC

(*cyclical redundancy check*) A method used to determine the validity of a communication transmission. The transmission contains a bit field that constitutes a checksum. The message is used to calculate the checksum by the transmitter according to the content of the message. Receiving nodes, then recalculate the field in the same manner. Any discrepancy in the value of the 2 CRC calculations indicates that the transmitted message and the received message are different.

## D

### DHCP

(*dynamic host configuration protocol*) An advanced extension of BOOTP. DHCP is more advanced, but both DHCP and BOOTP are common. (DHCP can handle BOOTP client requests.)

### 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.

**Ethernet**

A physical and data link layer technology for LANs, also known as IEEE 802.3.

**EtherNet/IP**

*(Ethernet industrial protocol)* An open communications protocol for manufacturing automation solutions in industrial systems. EtherNet/IP is in a family of networks that implement the common industrial protocol at its upper layers. The supporting organization (ODVA) specifies EtherNet/IP to accomplish global adaptability and media independence.

**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**

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

**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 within a project.

**H**

**hex**

*(hexadecimal)*

**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.

**IEEE 802.3**

A collection of IEEE standards defining the physical layer, and the media access control sublayer of the data link layer, of wired Ethernet.

**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.

**IP**

(*Internet protocol*) Part of the TCP/IP protocol family that tracks the Internet addresses of devices, routes outgoing messages, and recognizes incoming messages.

**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).

**LWORD**

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

**M****MAC address**

(*media access control address*) A unique 48-bit number associated with a specific piece of hardware. The MAC address is programmed into each network card or device when it is manufactured.

**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.

**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.

**protocol**

A convention or standard definition that controls or enables the connection, communication, and data transfer between 2 computing system and devices.

**R**

**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**

**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.

**string**

A variable that is a series of ASCII characters.

**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.

**TCP**

(*transmission control protocol*) A connection-based transport layer protocol that provides a simultaneous bi-directional transmission of data. TCP is part of the TCP/IP protocol suite.

**U****UDINT**

(*unsigned double integer*) Encoded in 32 bits.

**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.





## C

### cycle

- IsFirstMastColdCycle, *40*
- IsFirstMastCycle, *41*
- IsFirstMastWarmCycle, *43*

## D

### Data Types

- DataFileCopyError, *71*
  - DataFileCopyLocation, *72*
  - ETH\_R\_FRAME\_PROTOCOL, *75*
  - ETH\_R\_IP\_MODE, *76*
  - ETH\_R\_PORT\_DUPLEX\_STATUS, *77*
  - ETH\_R\_PORT\_IP\_STATUS, *78*
  - ETH\_R\_PORT\_LINK\_STATUS, *79*
  - ETH\_R\_PORT\_SPEED, *80*
  - ETH\_R\_RUN\_IDLE, *81*
  - ExecuteScriptError, *73*
  - PLC\_R\_APPLICATION\_ERROR, *59*
  - PLC\_R\_BOOT\_PROJECT\_STATUS, *61*
  - PLC\_R\_IO\_STATUS, *62*
  - PLC\_R\_SDCARD\_STATUS, *63*
  - PLC\_R\_STATUS, *64*
  - PLC\_R\_STOP\_CAUSE, *65*
  - PLC\_R\_TERMINAL\_PORT\_STATUS, *67*
  - PLC\_R\_TM3\_BUS\_STATE, *68*
  - PLC\_W\_COMMAND, *69*
  - RTCSETDRIFT\_ERROR, *87*
  - TM3\_BUS\_W\_IJOBUSERRMOD, *86*
  - TM3\_ERR\_CODE, *83*
  - TM3\_MODULE\_R\_ARRAY\_TYPE, *84*
  - TM3\_MODULE\_STATE, *85*
- ### DataFileCopy
- copying data to or from a file, *47*
- ### DataFileCopyError
- Data Types, *71*
- ### DataFileCopyLocation
- Data Types, *72*

## E

- ETH\_R
  - System Variable, *27*
- ETH\_R\_FRAME\_PROTOCOL
  - Data Types, *75*
- ETH\_R\_IP\_MODE
  - Data Types, *76*
- ETH\_R\_PORT\_DUPLEX\_STATUS
  - Data Types, *77*
- ETH\_R\_PORT\_LINK\_STATUS
  - Data Types, *79*
- ETH\_R\_PORT\_SPEED
  - Data Types, *80*
- ETH\_W
  - System Variable, *33*
- ExecuteScript
  - running script commands, *50*
- ExecuteScriptError
  - Data Types, *73*

## F

- file copy commands
  - DataFileCopy, *47*
- functions
  - differences between a function and a function block, *92*
  - how to use a function or a function block in IL language, *93*
  - how to use a function or a function block in ST language, *96*

## G

- GetRtc
  - getting real time clock (RTC) value, *39*

**I**

- IsFirstMastColdCycle
  - first cold start cycle, *40*
- IsFirstMastCycle
  - first mast cycle, *41*
- IsFirstMastWarmCycle
  - first warm start cycle, *43*

**M**

- M241 PLCSystem
  - DataFileCopy, *47*
  - ExecuteScript, *50*
  - GetRtc, *39*
  - IsFirstMastColdCycle, *40*
  - IsFirstMastCycle, *41*
  - IsFirstMastWarmCycle, *43*
  - SetRTCDrift, *44*
  - TM3\_GetModuleBusStatus, *53, 55*
  - TM3\_GetModuleFWVersion, *54*

**P**

- PLC\_R
  - System Variable, *18*
- PLC\_R\_APPLICATION\_ERROR
  - Data Types, *59*
- PLC\_R\_BOOT\_PROJECT\_STATUS
  - Data Types, *61*
- PLC\_R\_IO\_STATUS
  - Data Types, *62*
- PLC\_R\_SDCARD\_STATUS
  - Data Types, *63*
- PLC\_R\_STATUS
  - Data Types, *64*
- PLC\_R\_STOP\_CAUSE
  - Data Types, *65*
- PLC\_R\_TERMINAL\_PORT\_STATUS
  - Data Types, *67*
- PLC\_R\_TM3\_BUS\_STATE
  - Data Types, *68*
- PLC\_W
  - System Variable, *22*
- PLC\_W\_COMMAND
  - Data Types, *69*

- PROFIBUS\_R
  - System Variable, *36*

**R**

- real time clock
  - GetRtc, *39*
  - SetRTCDrift, *44*
- RTC
  - GetRtc, *39*
  - SetRTCDrift, *44*
- RTCSETDRIFT\_ERROR
  - Data Types, *87*

**S**

- script commands
  - ExecuteScript, *50*
- SERIAL\_R
  - System Variable, *24*
- SERIAL\_W
  - System Variable, *25*
- SetRTCDrift
  - accelerating or slowing the RTC frequency, *44*
  - System Variable
    - ETH\_R, *27*
    - ETH\_W, *33*
    - PLC\_R, *18*
    - PLC\_W, *22*
  - System variable
    - PROFIBUS\_R, *36*
  - System Variable
    - SERIAL\_R, *24*
    - SERIAL\_W, *25*
    - TM3\_BUS\_W, *35*
    - TM3\_MODULE\_R, *34*
  - System Variables
    - Definition, *13*
    - Using, *15*

**T**

- TM3 module bus status
  - TM3\_GetModuleBusStatus, *53*

- 
- TM3 module firmware version
    - TM3\_GetModuleFWVersion, *54*
  - TM3 module internal status
    - TM3\_GetModuleInternalStatus, *55*
  - TM3\_BUS\_W
    - system variable, *35*
  - TM3\_BUS\_W\_IOBUSERRMOD
    - Data Types, *86*
  - TM3\_ERR\_CODE
    - Data Types, *83*
  - TM3\_GetModuleBusStatus
    - getting the bus status of a TM3 module, *53*
  - TM3\_GetModuleFWVersion
    - getting the firmware version of a TM3 module, *54*
  - TM3\_GetModuleInternalStatus
    - getting the internal status of a TM3 module, *55*
  - TM3\_MODULE\_R
    - System Variable, *34*
  - TM3\_MODULE\_R\_ARRAY\_TYPE
    - Data Types, *84*
  - TM3\_MODULE\_STATE
    - Data Types, *85*

