Modicon M258 Logic Controller

System Functions and Variables M258 PLCSystem Library Guide

04/2017

EIO0000000584.08

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.

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

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

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

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

Table of Contents

Safety Information

Important Information

NOTICE

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

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

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

⚠ DANGER

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

⚠ WARNING

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

⚠ CAUTION

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

NOTICE

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

PLEASE NOTE

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

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

About the Book

At a Glance

Document Scope

This document will acquaint you with the system functions and variables offered within the Modicon M258 Logic Controller. The M258 PLCSystem library contains functions and variables to get information and send commands to the controller system.

This document describes the data types functions and variables of the M258 PLCSystem library.

The following basic knowledge is required:

- basic information on functionality, structure and configuration of the M258
- 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
Modicon M258 Logic Controller Programming Guide	*EIO0000000402 (Eng)* *EIO0000000403 (Fre)* *EIO0000000404 (Ger)* *EIO0000000405 (Spa)* *EIO0000000406 (Ita)* *EIO0000000407 (Chs)*

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.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

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

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

Standard	Description
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

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

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

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

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

Chapter 1
M258 System Variables

Overview

This chapter:

- gives an introduction to the system variables *(see page 14)*
- describes the system variables *(see page 20)* included with the M258 PLCSystem library

What Is in This Chapter?

This chapter contains the following sections:

Section 1.1
System Variables: Definition and Use

Overview

This section defines system variables and how to implement them in the Modicon M258 Logic Controller.

What Is in This Section?

This section contains the following topics:

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
  - %MW62000 to %MW62199 for read/write 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

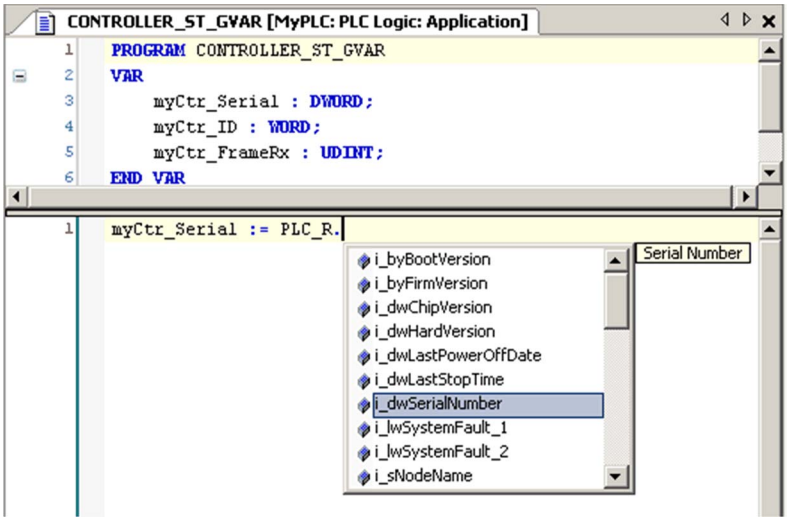
This section describes the steps required to program and to use system variables in SoMachine.

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:

PLC_R: Controller Read-Only System Variables

Variable Structure

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

Modbus Address (1)	Var Name	Type	Comment
60000	i_wVendorID	WORD	Controller Vendor ID. 101A hex = Schneider Electric
60001	i_wProductID	WORD	Controller Reference ID. **NOTE:** Vendor ID and Reference ID are the components of the Target ID of the controller displayed in the communication settings view (Target ID = 101A XXXX hex).
60002	i_dwSerialNumber	DWORD	Controller Serial Number
60004	i_byFirmVersion[0..3]	ARRAY[0..3] OF BYTE	Controller Firmware Version [aa.bb.cc.dd]: • i_byFirmVersion[0] = aa • ... • i_byFirmVersion[3] = dd
60006	i_byBootVersion[0..3]	ARRAY[0..3] OF BYTE	Controller Boot Version [aa.bb.cc.dd]: • i_byBootVersion[0] = aa • ... • i_byBootVersion[3] = dd
60008	i_dwHardVersion	DWORD	Controller Hardware Version.
60010	i_dwChipVersion	DWORD	Controller Coprocessor Version.
60012	i_wStatus	PLC_R_STATUS *(see page 65)*	State of the controller.
60013	i_wBootProjectStatus	PLC_R_BOOT_PROJECT_STATUS *(see page 63)*	Returns information about the boot application stored in FLASH memory.
60014	i_wLastStopCause	PLC_R_STOP_CAUSE *(see page 66)*	Cause of the last transition from RUN to another state.
60015	i_wLastApplicationError	PLC_R_APPLICATION_ERROR *(see page 61)*	Cause of the last controller exception.

Modbus Address (1)	Var Name	Type	Comment
60016	i_lwSystemFault_1	LWORD	Bit field FFFF FFFF FFFF FFFF hex indicates no error detected. A bit at low level means that an error has been detected: ● bit 0 = Embedded Expert error detected. See i_wIOStatus1 for diagnostic ● bit 1 = TM5 I/O error detected. See i_wIOStatus2 for diagnostic ● bit 2 = The Ethernet 0 error is not detected when you are in BOOTP or DHCP without Master. ● bit 3 = Serial 0 error detected ● bit 4 = CAN 0 error detected ● bit 5 = CAN 1 error detected ● bit 6 = Interface bus Module 0 error detected ● bit 7 = Interface bus Module 1 error detected **NOTE:** After a power cycle, the SystemFault can be in an error state during several cycles.
60020	i_lwSystemFault_2	LWORD	Not used.
60024	i_wIOStatus1	PLC_R_IO_STATUS *(see page 64)*	Embedded Expert I/O status.
60025	i_wIOStatus2	PLC_R_IO_STATUS *(see page 64)*	TM5 I/O status.
60026	i_wClockBatterystatus	WORD	Status of the Real Time Clock battery charge: ● 0000 hex = Battery charge is low ● FFFF hex = Battery charge is correct
60028	i_dwAppliSignature1	DWORD	First DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
60030	i_dwAppliSignature2	DWORD	Second DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.

Modbus Address (1)	Var Name	Type	Comment
60032	i_dwAppliSignature3	DWORD	Third DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
60034	i_dwAppliSignature4	DWORD	Fourth DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
(1) Not accessible through the application.			

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_dwiLastStopTime	DWORD	The time of the last detected STOP in seconds beginning with January 1, 1970 at 00:00.
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. **NOTE:** Convert this value into date and time by using the function SysTimeRtcConvertUtcToDate. For more information about Time and Date conversion, refer to the Systime Library Guide *(see SoMachine, Getting & Setting Real Time Clock, SysTimeRtc and SysTimeCore Library Guide)*.
n/a	i_uiEventsCounter	UINT	Number of external events detected on inputs configured for external event detection since the last cold start. Reset by a Cold Start or by the PLC_W.q_wResetCounterEvent command.
n/a	i_wTerminalPortStatus	PLC_R_TERMINAL_PORT_STATUS *(see page 68)*	Status of the USB Programming Port (USB Mini-B).
n/a	i_wUSBHostStatus	PLC_R_USB_HOST_STATUS *(see page 69)*	Status of the USB Host port (USB A).

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_uiTM5BusState	UINT	TM5 bus state BitField: ● bits 0..3 = not used ● bit 4 = TM5 bus driver available ● bit 5 = TM5 bus hardware found ● bit 6 = TM5 bus configuration done successfully ● bits 7 = TM5 bus is operational ● bits 8 = not used ● bit 9 = error detected during TM5 bus configuration ● bit 10..15 = not used
n/a	i_uiTM5SyncErrCnt	UINT	Number of invalid synchronous frames detected on TM5 bus. Reset with the PLC_W.q_wResetTM5Counters command and at Power Off.
n/a	i_uiTM5AsynErrCnt	UINT	Number of invalid asynchronous frames detected on TM5 bus. Reset with the PLC_W.q_wResetTM5Counters command and at Power Off.
n/a	i_uiTM5BreakCnt	UINT	Number of TM5 bus resets detected. Reset with the PLC_W.q_wResetTM5Counters command and at Power Off.
n/a	i_uiTM5TopoChangedCnt	UINT	Number of changes in the TM5 bus topology. Reset with the PLC_W.q_wResetTM5Counters command and at Power Off.
n/a	i_uiTM5BusCycleCnt	UINT	Number of TM5 bus cycles from Cold Start. Reset with the PLC_W.q_wResetTM5Counters command and at Power Off.
n/a	i_wTM5BrokendownSlot	WORD	00..FE hex = Slot number of an inoperative TM5 module. FF hex = All TM5 modules report they are operational.

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 0 to this variable 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 *(see page 70)*	Controller RUN / STOP command executed when the system variable PLC_R.q_uiOpenPLCControl value passes from 0 to 6699.
n/a	q_wResetTM5counters	WORD	Transition from 0 to 1 resets all TM5 counters of PLC_R Structured System Variables (PLC_R.i_uiTM5SyncErrCnt to PLC_R.i_uiTM5BusCycleCnt) To reset the counters again, it is necessary to write 0 to this variable before another transition from 0 to 1 can take place.

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:

SERIAL_R[0...2]: Serial Line Read-Only System Variables

Introduction

For the M258 Logic Controller:

- `Serial_R[0]` refers to the embedded serial line
- `Serial_R[1]` refers to the optional serial line PCI module, if installed
- `Serial_R[2]` refers to the optional serial line PCI module, if installed

Variable Structure

This table describes the parameters of the `SERIAL_R[0...2]` system variables:

%MW	Var Name	Type	Comment
Serial Line			
n/a	`i_udiFramesTransmittedOK`	`UDINT`	Number of frames successfully transmitted.
n/a	`i_udiFramesReceivedOK`	`UDINT`	Number of frames received without any errors detected.
n/a	`i_udiRX_MessagesError`	`UDINT`	Number of frames received with errors detected (checksum, parity).
Modbus Specific			
n/a	`i_uiSlaveExceptionCount`	`UINT`	Number of Modbus exception responses returned by the logic controller.
n/a	`i_udiSlaveMsgCount`	`UINT`	Number of messages received from the Master and addressed to the logic controller.
n/a	`i_uiSlaveNoRespCount`	`UINT`	Number of Modbus broadcast requests received by the logic controller.
n/a	`i_uiSlaveNakCount`	`UINT`	Not used
n/a	`i_uiSlaveBusyCount`	`UINT`	Not used
n/a	`i_uiCharOverrunCount`	`UINT`	Number of character overruns.

n/a means that there is no predefined %MW mapping for this system variable.
Not used means that the variable is not maintained by the system, and that if the value of the variable is non-zero, it should be considered extraneous.

The `SERIAL_R` counters are reset on:

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

SERIAL_W[0...2]: Serial Line Read/Write System Variables

Introduction

For the M258 Logic Controller:

- `Serial_W[0]` refers to the embedded serial line
- `Serial_W[1]` refers to the PCI serial line
- `Serial_W[2]` refers to the PCI serial line

Variable Structure

This table describes the parameters of the `SERIAL_W[0...2]` system variable:

%MW	Var Name	Type	Comment
n/a	`q_wResetCounter`	`WORD`	Transition from 0 to 1 resets all counters. To reset the counters again, it is necessary to write 0 to this variable 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:

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[0..3]	ARRAY[0..3] OF BYTE	IP address [aaa.bbb.ccc.ddd]: • i_byIPAddress[0] = aaa • ... • i_byIPAddress[3] = ddd
60052	i_bySubNetMask[0..3]	ARRAY[0..3] OF BYTE	Subnet Mask [aaa.bbb.ccc.ddd]: • i_bySub-netMask[0] = aaa • ... • i_bySub-netMask[3] = ddd
60054	i_byGateway[0..3]	ARRAY[0..3] OF BYTE	Gateway address [aaa.bbb.ccc.ddd]: • i_byGateway[0] = aaa • ... • i_byGateway[3] = ddd
60056	i_byMACAddress[0..5]	ARRAY[0..5] OF BYTE	MAC address [aa.bb.cc.dd.ee.ff]: • i_byMACAddress[0] = aa • ... • i_byMACAddress[5] = ff
60059	i_sDeviceName	STRING(16)	Name used to get IP address from server.
n/a	i_wIpMode	ETH_R_IP_MODE *(see page 74)*	Method used to obtain an IP address.
n/a	i_byFDRServerIPAddress[0..3]	ARRAY[0..3] OF BYTE	The IP address [aaa.bbb.ccc.ddd] of the DHCP or BootP server: • i_byFDRServerIPAddress [0] = aaa • ... • i_byFDRServerIPAddress [3] = ddd Equals 0.0.0.0 if Stored IP or Default IP used.
n/a	i_udiOpenTcpConnections	UDINT	Number of open TCP connections.
n/a means that there is no predefined %MW mapping for this system variable.			

%MW	Var Name	Type	Comment
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	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_wPortALinkStatus	ETH_R_PORT_LINK_STATUS *(see page 77)*	Link of the Ethernet Port (0 = No Link, 1 = Link connected to another Ethernet device).
n/a	i_wPortASpeed	ETH_R_PORT_SPEED *(see page 78)*	Ethernet Port network speed (10Mb/s or 100Mb/s).
n/a	i_wPortADuplexStatus	ETH_R_PORT_DUPLEX_STATUS *(see page 76)*	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.
n/a	i_wPortAIpStatus	ETH_R_PORT_IP_STATUS *(see page 79)*	Ethernet TCP/IP port stack status.
n/a means that there is no predefined %MW mapping for this system variable.			

%MW	Var Name	Type	Comment
Modbus TCP/IP Specific			
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.
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 means that there is no predefined %MW mapping for this system variable.			

%MW	Var Name	Type	Comment
EtherNet/IP Specific			
n/a	i_udiETHIP_IOMessagingTransmitted	UDINT	EtherNet/IP Class 1 frames transmitted. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
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 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_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	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 80)*	Run (value = 1)/Idle(value = 0) flag for EtherNet/IP class 1 connection.

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.

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 0 to this variable 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
TM5_MODULE_R Structure

TM5_MODULE_R[1..254]: TM5 Modules Read Only System Variables

Introduction

`TM5_MODULE_R` is an array of 254 `TM5_MODULE_R_STRUCT` type. Each element of the array returns diagnostic **System Variables** for the corresponding TM5 Module.

For M258:

- `TM5_MODULE_R[1]` refer to the TM5 Module 1
- ...
- `TM5_MODULE_R[254]` refer to the TM5 Module 254

Variable Structure

The following table describes the parameters of the `TM5_MODULE_R[1..254]` System Variable:

%MW	Var Name	Type	Comment
n/a	`i_wVendorID`	`WORD`	TM5 Module Vendor ID of the target.
n/a	`i_wProductID`	`WORD`	TM5 Module Type ID of the target.
n/a	`i_dwSerialNumber`	`DWORD`	TM5 Module Serial Number.
n/a	`i_wFirmVersion`	`WORD`	TM5 Module Firmware version.
n/a	`i_wBootVersion`	`WORD`	TM5 Module Boot version.
n/a	`i_wModuleState`	`TM5_MODULE_STATE` *(see page 81)*	Describes the state of the TM5 module. The module is operational when `TM5_ACTIVE` is returned.

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

Chapter 2
M258 System Functions

Overview

This chapter describes the system functions included in the M258 PLCSystem library.

What Is in This Chapter?

This chapter contains the following sections:

Section 2.1
M258 Read Functions

Overview

This section describes the read functions included in the M258 PLCSystem library.

What Is in This Section?

This section contains the following topics:

DM72FGetImmediateInput: Read Input of an Embedded Expert I/O

Function Description

This function is applicable to the **Embedded Expert I/O Block** DM72F0 and DM72F1. It returns the current physical value of the input, which may be different from the current logical value of that input. The value of the variable for that input does not change until the next bus cycle.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 95)*.

I/O Variable Description

The following table describes the input variables:

Input	Type	Comment
`Block`	`INT`	Targeted block: • 0= DM72F0 • 1= DM72F1
`Input`	`INT`	Targeted input of the block. 0..6= DI0..DI6

The following table describes the output variable:

Output	Type	Comment
`DM72FGetImmediateInput`	`BOOL`	Value of the input `<Input>` of the block `<block>`= FALSE/TRUE.

The following table describes the input/output variables:

Input/Output	Type	Comment
Error	BOOL	FALSE= operation is correct. TRUE= detected operation error, the function returns an invalid value.
ErrID	IMMEDIATE_FUNC_ERR_TYPE	Detected operation error code when Error is TRUE.

getTM5Delay: Number of TM5 Bus Cycles Without Valid Exchange

Function Description

This function will return the number of TM5 bus cycles without valid exchange with a targeted TM5 module.

NOTE: For TM5 module diagnostic, see System Variable TM5_MODULE_R *(see page 34)*.

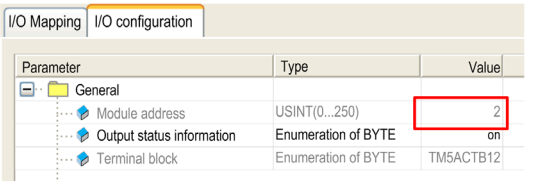
Graphical Representation

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 95)*.

I/O Variable Description

The following table describes the input variable:

Input	Type	Comment
NodeId	DINT	Targeted TM5 Module **Module address** (to get the **Module address**, verify the value in the **I/O Configuration** tab). 

The following table describes the output variable:

Output	Type	Comment
getTM5Delay	USINT	The variable can take the following values: ● 0= OK ● [1..3]= 1 to 3 cycles without valid exchange ● -1= more than 3 cycles without valid exchange or invalid parameter

Example

The following example describes how to get the delay of the first TM5 Module:

```
VAR
    delay : DINT;
    //Slot ID is 1 for the first TM5 module
            slot_ID : USINT := 1;
END_VAR
```

```
delay = getTM5Delay(slot_ID);
```

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 95)*.

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 42)*.

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 95)*.

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 95)*.

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 42)*.

Section 2.2
M258 Write Functions

Overview

This section describes the write functions included in the M258 PLCSystem library.

What Is in This Section?

This section contains the following topics:

DM72F•SetImmediateOutput•: Write Output of an Embedded Expert I/O

Function Description

This function sets the current physical value of the fast output of an embedded expert I/O (DM72F•).

There is a function for each fast output:

- `DM72F0SetImmediateOutput0`
- `DM72F0SetImmediateOutput1`
- `DM72F1SetImmediateOutput0`
- `DM72F1SetImmediateOutput1`

NOTE: Output forcing *(see Modicon M258 Logic Controller, Programming Guide)* overrides all other commands to an output irrespective of the task programming that is being executed.

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 95)*.

I/O Variable Description

The following table describes the input variable:

Input	Type	Comment
`Value`	`BOOL`	Requested output value.

The following table describes the output variable:

Output	Type	Comment
`DM72F`**b**[1]`SetImmediateOutput`**n**[2]	`BOOL`	TRUE = the physical output value is set.
(1) **b** is the targeted block: • 0= `DM72F0` • 1= `DM72F1` (2) **n** is the targeted output of the block: • 0= `DO0` • 1= `DO1`		

Implementing DM72F•SetImmediateOutput•

SoMachine returns a compilation detected error for the following cases:

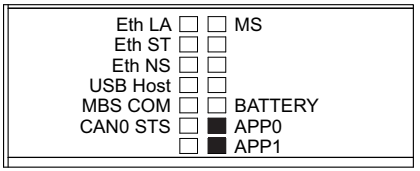
- The function DM72F•SetImmediateOutput• is used in more than one task.
- The %Q associated to DM72F•SetImmediateOutput• is used in the application.
- The output is already dedicated to an **Embedded Expert I/O Block** function (for example: PWM, frequency generator, encoder's reflex output, alarm).

SetLEDBehaviour: Determines the Behavior of a LED

Function Description

This function controls the diagnostic LEDs APP0 and APP1.

The following figure shows the LEDs on the front panel display:



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 95)*.

I/O Variables Description

The following table describes the input parameters:

Inputs	Type	Comment
LedId	LED_ID *(see page 84)*	ID of the application LED.
LedColor	LED_COLOR *(see page 87)*	Color of the application LED.
LedBhv	LED_BHV *(see page 85)*	Mode of the application LED.

The following table describes the output variable:

Output	Type	Comment
SetLEDBehaviour	LED_BHV_ERROR *(see page 86)*	Returns NO_ERROR (00 hex) if command is correct otherwise returns the ID code of the detected error.

Example

This example shows how to command LED APP0 to illuminate green:

```
VAR
    myLEDStatus : LED_BHV_ERROR;
    myLED : LED_ID := LED_0;
    myLEDColor : LED_COLOR := LED_GREEN;
    myLEDMode : LED_BHV := LED_ON;
END_VAR
```

```
myLEDStatus := SetLedBehaviour(myLED, myLEDColor, myLEDMode);
```

NOTE: LED colors are controlled separately and can be mixed, therefor turn off the current color before lighting the new one. The table below shows an example of `SetLedBehaviour` commands sequence with associated LED behaviour:

step	LedId	LedColor	LedBhv	GREEN flashing mode	RED flashing mode
1	LED_0	-	-	OFF	OFF
2	LED_0	LED_GREEN	LED_ON	ON	OFF
3	LED_0	LED_GREEN	LED_OFF	OFF	OFF
4	LED_0	LED_RED	LED_ON	OFF	ON

SetRTCDrift: Adjust the Real Time Clock Each Week

Function Description

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

NOTE: The `SetRTCDrift` function needs to be programmed to be executed only during the First Mast Cycle.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 95)*.

I/O Variables Description

This table describes the input parameters:

Inputs	Type	Comment
`RtcDrift`	`SINT(-29...+29)`	Correction in seconds (-29 ... +29)
`Day`	DAY_OF_WEEK *(see page 89)*	Day of change.
`Hour`	HOUR *(see page 90)*	Hour of change.
`Minute`	MINUTE *(see page 91)*	Minute of change.

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

This table describes the output variable:

Output	Type	Comment
`SetRTCDrift`	RTCSETDRIFT_ERROR *(see page 88)*	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 and 20 seconds is added to the RTC every Tuesday at 5:45

```
VAR
    MyRTCDrift : SINT (-29...+29) := 0;
    MyDay : DAY_OF_WEEK;
    MyHour : HOUR;
    MyMinute : MINUTE;
END_VAR
```

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

Section 2.3
M258 User Functions

Overview

This section describes the `DataFileCopy` and `ExecuteScript` functions included in the M258 PLCSystem library.

What Is in This Section?

This section contains the following topics:

DataFileCopy: Copy File Commands

Function Block Description

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

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 M258 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 95)*.

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 *.DTA* 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 */usr/DTA* in internal file system. 1: the file location is */usr/DTA* in external file system (USB key).

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 *(see page 71)*	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 USB script commands:

- Download
- Upload
- SetNodeName
- Delete
- Reboot

Refer to Script and Files Generation with USB Mass Storage *(see Modicon M258 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 95)*.

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	USB script command syntax. Simultaneous command executions are not allowed: if a command is being executed from another function block or from USB script then the function block queues the command and does not execute it immediately. **NOTE:** A USB script executed from a USB memory key is considered as being executed until the USB key 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 72)*	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=> );
```

Chapter 3
M258 PLCSystem Library Data Types

Overview

This chapter describes the data types of the M258 PLCSystem Library.

There are 2 kinds of data types available:

- System variable data types are used by the system variables *(see page 13)* of the M258 PLCSystem Library (PLC_R, PLC_W,...).
- System function data types are used by the read/write system functions *(see page 35)* of the M258 PLCSystem Library.

What Is in This Chapter?

This chapter contains the following sections:

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:

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: 1. **Build → Clean All** 2. Export/Import your application. 3. Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_UNRESOLVED_EXTREFS	0018 hex	Undefined functions detected.	Delete the unresolved functions from the application.
PLC_R_APP_ERR_IEC_TASK_CONFIG_ERROR	0025 hex	Incorrect Task configuration parameters detected.	Your application might be corrupted. To resolve this issue, use one of the methods: 1. **Build → Clean All** 2. Export/Import your application. 3. Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_ILLEGAL_INSTRUCTION	0050 hex	Undefined instruction detected.	Debug your application to resolve the problem.
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_TOO_MANY_EVENT	4E20 hex	External I/O events rate is too high.	Reduce expert I/O event frequency.

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. If rearming mode is manual, PLC_R_IO_STATUS is set to PLC_R_IO_SHORTCUT_FAULT when power supply recovers.
`PLC_R_IO_POWER_SUPPLY_FAULT`	0004 hex	Inputs/Outputs power supply error detected.

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 *programming guide*).
`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, USB key, or `PLC_W command.`	–
`PLC_R_STOP_FROM_ INPUT`	0B hex	Stop required by a controller input.	–
`PLC_R_STOP_REASON_RETAIN_MISMATCH`	0C hex	Stopped after an unsuccessful check context test during rebooting.	Retain variables were deleted because they were not referenced in the application. If the application sets retain variables to their intialization values, the is available.

Enumerator	Value	Comment	What to do
PLC_R_STOP_REASON_BOOT_APPLI_MISMATCH	0D hex	Stopped after an unsuccessful compare between the boot application and the application that was in the memory before rebooting.	Create a valid boot application.
PLC_R_STOP_REASON_POWERFAIL	0E hex	Stopped after a power interruption.	–

For more information for reasons the controller has stopped, refer to the Controller State Description *(see Modicon M258 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_USB_HOST_STATUS: USB Host Port Connection Status Codes

Enumerated Type Description

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

Enumerator	Value	Comment
`USB_NOT_CONNECTED`	00 hex	No device (Memory Key) connected to the USB Host Port.
`USB_CONNECTION_IN_PROGRESS`	01 hex	Connection in progress(1).
`USB_CONNECTED`	02 hex	USB Host port connected to a device (Memory Key).
`USB_ERROR`	0F hex	Error detected during connection.

(1) Supported USB memory keys meet the following requirements:
- 1 Gbytes minimum capacity
- USB 2.0 specification or less
- FAT16 or FAT32 file system
- A volume label must be set

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

DataFileCopyError: Detected Error Codes

Enumerated Type Description

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

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

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
`CMD_OK`	00 hex	No error detected.
`ERR_CMD_UNKNOWN`	01 hex	The command is not recognized.
`ERR_USB_KEY_MISSING`	02 hex	USB key is not present.
`ERR_SEE_FWLOG`	03 hex	There was an error detected during command execution, see `FwLog.txt`. For more information, refer to File Type *(see Modicon M258 Logic Controller, Programming Guide)*.
`ERR_ONLY_ONE_COMMAND_ALLOWED`	04 hex	An attempt was made to execute several scripts simultaneously.
`CMD_BEING_EXECUTED`	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:

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_FRAME_PROTOCOL: Frame Transmission Protocol Codes

Enumerated Type Description

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

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

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_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 from server to device.
`ETH_R_LINK_UP`	01 hex	Communication link from device to server.

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_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_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_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
TM5_MODULE_R/W System Variables Data Types

TM5_MODULE_STATE: TM5 Expansion Module Status Codes

Enumerated Type Description

The `TM5_MODULE_STATE` enumeration data type contains the following values:

Enumerator	Value	Comment
`TM5_INACTIVE`	00 hex	State machine inactive.
`TM5_BOOT`	0A hex	Boot in progress.
`TM5_FWDNLD`	0B hex	Firmware download in progress.
`TM5_PREOP`	14 hex	Basic initialization.
`TM5_OPERATE`	1E hex	Register initialization.
`TM5_ACTIVE`	64 hex	Module communication is active.
`TM5_ERROR`	C8 hex	Module in Detected Error state.
`TM5_UNSUP`	C9 hex	TM5 Module not supported.
`TM5_NOCFG`	CA hex	No configuration available.

Section 3.6
PROFIBUS_R System Variables Data Types

PROFIBUS_R: Profibus Diagnostic System Variable

Description

The table lists the parameters of the `PROFIBUS_R` system variable (`PROFIBUS_R_STRUCT`) type:

Variable Name	Type	Value	Comment
`i_wPNOIdentifier`	WORD	0x0D73	Profibus slave identification number
`i_wBusAdr`	UINT	1...126	Profibus slave address number The address 126 is reserved.
`i_CommState`	UDINT	0 - unknown 1 - not configured 2 - stop 3 - idle 4 - operate	Profibus status operate mode: cyclic I/Os are running
`i_CommError`	UDINT	0 - no error <>0 error code	communication error detected by the Profibus module, indicated by an error code *(see Modicon TM5, PCI Modules Configuration, Programming Guide)*
`i_ErrorCount`	UDINT	≥0	number of communication errors that have been detected

Section 3.7
System Function Data Types

Overview

This section describes the different system function data types of the M258 PLCSystem library.

What Is in This Section?

This section contains the following topics:

LED_ID: SetLEDBehaviour Function LedId Parameter Codes

Enumerated Type Description

The LED_ID enumeration data type contains the following values:

Enumerator	Value	Comment
LED_0	00 hex	Identifier for application LED APP0.
LED_1	01 hex	Identifier for application LED APP1.

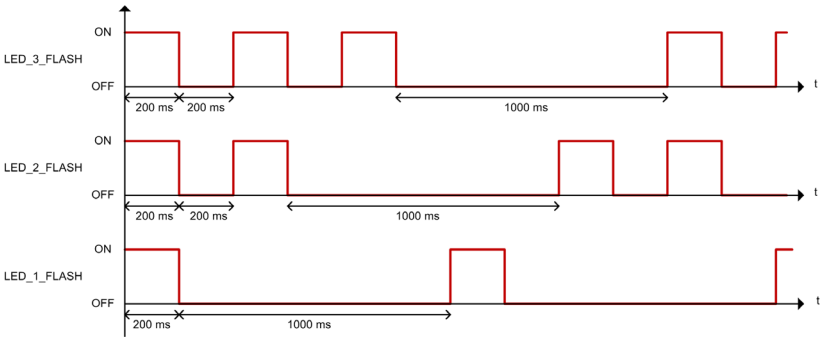
LED_BHV: SetLEDBehaviour Function LedBhv Parameter Codes

Enumerated Type Description

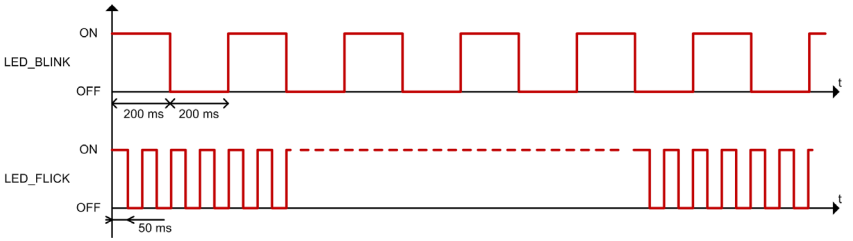
The LED_BHV enumeration data type contains the following values:

Enumerator	Value	Comment
LED_3_FLASH	-3 dec	LED is flashing in mode 3 (see diagram below).
LED_2_FLASH	-2 dec	LED is flashing in mode 2 (see diagram below).
LED_1_FLASH	-1 dec	LED is flashing in mode 1 (see diagram below).
LED_OFF	0 dec	LED is permanently off.
LED_ON	1 dec	LED is permanently on.
LED_BLINK	2 dec	LED is flashing at 2.5 Hz (see diagram below).
LED_FLICK	3 dec	LED is flashing at 10 Hz (see diagram below).

The clock diagram below describes the Application LEDs flashing modes LED_x_FLASH:



The clock diagram below describes the Application LEDs flashing modes LED_BLINK and LED_FLICK:



LED_BHV_ERROR: Detected `SetLEDBehaviour` Function Error Codes

Enumerated Type Description

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

Enumerator	Value	Comment
`NO_ERROR`	00 hex	LED behavior setting function executed without detected error.
`UNKNOWN_LED`	01 hex	LED_ID parameter is unknown.
`UNKNOWN_COLOR`	02 hex	LED_COLOR parameter is unknown.
`UNKNOWN_STATE`	03 hex	LED status, as contained in LED_BHV parameter is unknown.
`FIRMWARE_ERROR`	04 hex	Command rejected by FW on detected error.

LED_COLOR: SetLEDBehaviour Function LedColor Parameter Codes

Enumerated Type Description

The LED_COLOR enumeration data type contains the following values:

Enumerator	Value	Comment
LED_RED	00 hex	LED color is red.
LED_GREEN	01 hex	LED color is green.

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	Day parameter unknown.
`RTC_BAD_HOUR`	02 hex	Hour parameter unknown.
`RTC_BAD_MINUTE`	03 hex	Minute parameter unknown.
`RTC_BAD_DRIFT`	04 hex	RTC Drift parameter out of range.
`RTC_INTERNAL_ERROR`	05 hex	RTC Drift settings rejected on internal error detected.

DAY_OF_WEEK: `SetRTCDrift` Function Day Parameter Codes

Enumerated Type Description

The enumeration data type contains the following values:

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

HOUR: `SetRTCDrift` Function Hour Parameter Type

Data Type Description

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

MINUTE: `SetRTCDrift` Function Minute Parameter Type

Data Type Description

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

Appendices

Appendix A
Function and Function Block Representation

Overview

Each function can be represented in the following languages:

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

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

What Is in This Chapter?

This chapter contains the following topics:

Differences Between a Function and a Function Block

Function

A function:

- is a POU (Program Organization Unit) that returns one immediate result.
- is directly called with its name (not through an instance).
- has no persistent state from one call to the other.
- can be used as an operand in other expressions.

Examples: boolean operators (`AND`), calculations, conversion (`BYTE_TO_INT`)

Function Block

A function block:

- is a POU (Program Organization Unit) that returns one or more outputs.
- needs to be called by an instance (function block copy with dedicated name and variables).
- each instance has a persistent state (outputs and internal variables) from one call to the other from a function block or a program.

Examples: timers, counters

In the example, `Timer_ON` is an instance of the function block `TON`:

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

How to Use a Function or a Function Block in IL Language

General Information

This part explains how to implement a function and a function block in IL language.

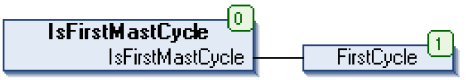
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

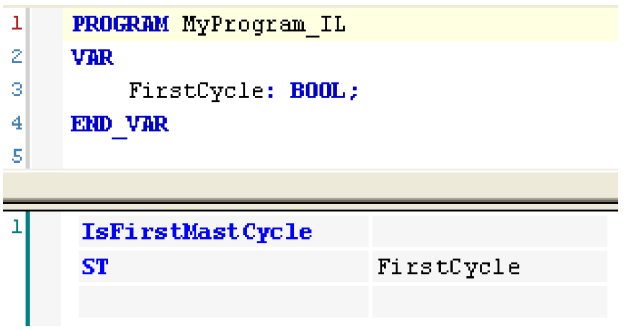
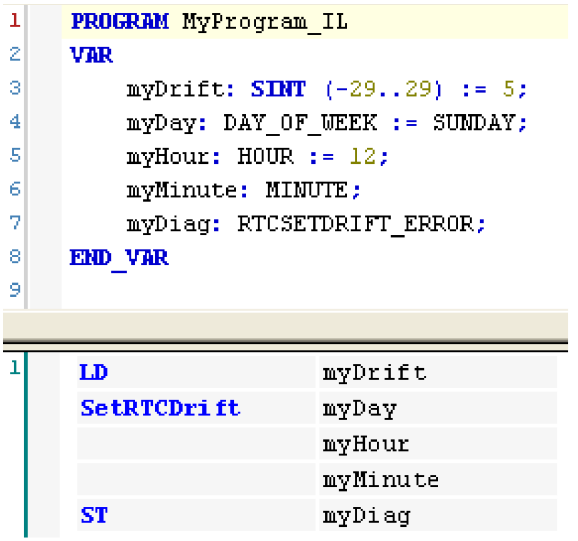
This procedure describes how to insert a function in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	If the function has 1 or more inputs, start loading the first input using LD instruction.
4	Insert a new line below and: ● type the name of the function in the operator column (left field), or ● use the **Input Assistant** to select the function (select **Insert Box** in the context menu).
5	If the function has more than 1 input and when Input Assistant is used, the necessary number of lines is automatically created with `???` in the fields on the right. Replace the `???` with the appropriate value or variable that corresponds to the order of inputs.
6	Insert a new line to store the result of the function into the appropriate variable: type ST instruction in the operator column (left field) and the variable name in the field on the right.

To illustrate the procedure, consider the Functions `IsFirstMastCycle` (without input parameter) and `SetRTCDrift` (with input parameters) graphically presented below:

Function	Graphical Representation
without input parameter: `IsFirstMastCycle`	
with input parameters: `SetRTCDrift`	

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

Function	Representation in POU IL Editor
IL example of a function without input parameter: `IsFirstMastCycle`	
IL example of a function with input parameters: `SetRTCDrift`	

Using a Function Block in IL Language

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

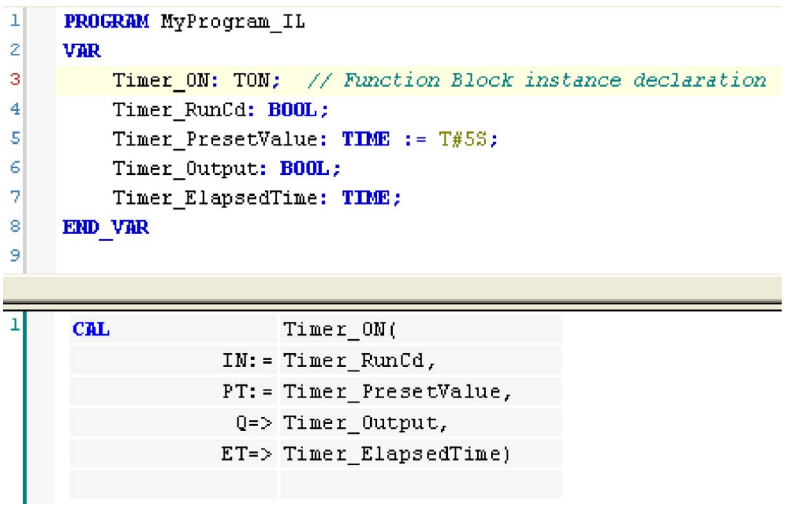
Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function block requires, including the instance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

Step	Action
3	Function Blocks are called using a CAL instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the CAL instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values to inputs are set by ":=". ● Values to outputs are set by "=>".
4	In the CAL right-side field, replace ??? with the instance name.
5	Replace other ??? with an appropriate variable or immediate value.

To illustrate the procedure, consider this example with the TON Function Block graphically presented below:

Function Block	Graphical Representation
TON	

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

Function Block	Representation in POU IL Editor
TON	

How to Use a Function or a Function Block in ST Language

General Information

This part explains how to implement a Function and a Function Block in ST language.

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

Using a Function in ST Language

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

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	Use the general syntax in the **POU ST Editor** for the ST language of a function. The general syntax is: `FunctionResult:= FunctionName(VarInput1, VarInput2,.. VarInputx);`

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

Function	Graphical Representation
`SetRTCDrift`	

The ST language of this function is the following:

Function	Representation in POU ST Editor

`SetRTCDrift`	`PROGRAM MyProgram_ST`
	`VAR myDrift: SINT(-29..29) := 5;`
	`myDay: DAY_OF_WEEK := SUNDAY;`
	`myHour: HOUR := 12;`
	`myMinute: MINUTE;`
	`myRTCAdjust: RTCDRIFT_ERROR;`
	`END_VAR`
	`myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);`

Using a Function Block in ST Language

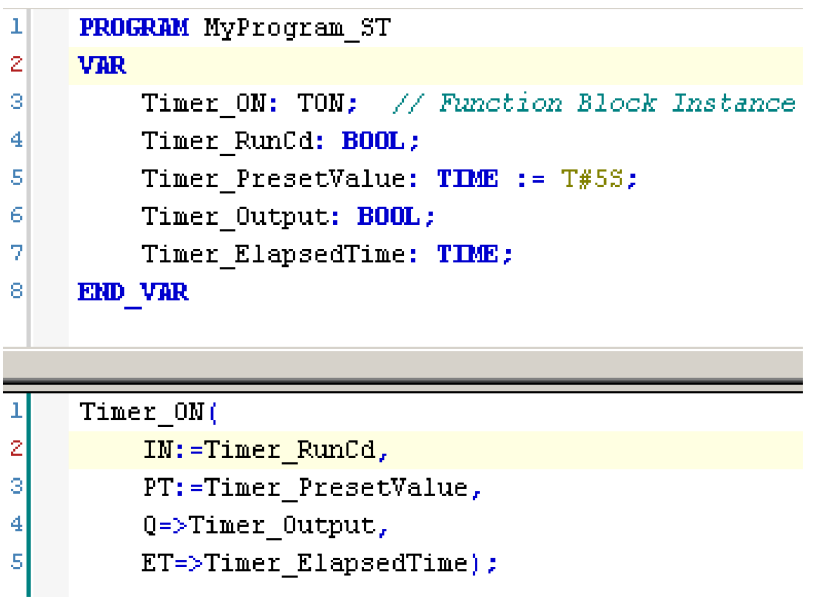
This procedure describes how to insert a function block in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation *(see SoMachine, Programming Guide)*.
2	Create the input and output variables and the instance required for the function block: ● Input variables are the input parameters required by the function block ● Output variables receive the value returned by the function block
3	Use the general syntax in the **POU ST Editor** for the ST language of a Function Block. The general syntax is: `FunctionBlock_InstanceName(Input1:=VarInput1, Input2:=VarInput2,... Ouput1=>VarOutput1, Ouput2=>VarOutput2,...);`

To illustrate the procedure, consider this example with the `TON` function block graphically presented below:

Function Block	Graphical Representation
`TON`	

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

Function Block	Representation in POU ST Editor
TON	

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

Glossary

!

%

According to the IEC standard, % is a prefix that identifies internal memory addresses in the logic controller to store the value of program variables, constants, I/O, and so on.

%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

CAN

(*controller area network*) A protocol (ISO 11898) for serial bus networks, designed for the interconnection of smart devices (from multiple manufacturers) in smart systems and for real-time industrial applications. Originally developed for use in automobiles, CAN is now used in a variety of industrial automation control environments.

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

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).

LED

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

LWORD

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

M

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

PCI

(*peripheral component interconnect*) An industry-standard bus for attaching peripherals.

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. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

program

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

protocol

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

PWM

(*pulse width modulation*) A fast output that oscillates between off and on in an adjustable duty cycle, producing a rectangular wave form (though you can adjust it to produce a square wave).

R

reflex output

Among the outputs of HSC are the reflex outputs associated to a threshold value that is compared to the counter value depending on the configuration of the HSC. The reflex outputs switch to either on or off depending on the configured relationship with the threshold.

RTC

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

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.

Index

T