Magelis SCU
HMI Controller
PLCSystem Library Guide

12/2016

EIO0000001246.03

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 HMI SCU controller. The HMI SCU 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 HMI SCU PLCSystem library.

The following basic knowledge is required:

- basic information on functionality, structure, and configuration of the HMI SCU
- 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 V4.2.

Related Documents

Title of Documentation	Reference Number
Magelis SCU HMI Controller Programming Guide	EIO0000001240 (eng), EIO0000001241 (fre), EIO0000001242 (ger), EIO0000001243 (spa), EIO0000001244 (ita), EIO0000001245 (chs)
Magelis SCU HMI Controller HSC Library Guide	EIO0000001512 (eng), EIO0000001513 (fre), EIO0000001514 (ger), EIO0000001515 (spa), EIO0000001516 (ita), EIO0000001517 (chs)
Magelis SCU HMI Controller PTO/PWM Library Guide	EIO0000001518 (eng), EIO0000001519 (fre), EIO0000001520 (ger), EIO0000001521 (spa), EIO0000001522 (ita), EIO0000001523 (chs)

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

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

Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

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

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

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

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

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

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

Chapter 1
HMI SCU System Variables

Overview

This chapter:

- provides an introduction to the System Variables *(see page 12)*
- describes the System Variables *(see page 21)* included with the HMI SCU 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 Magelis SCU HMI Controller.

What Is in This Section?

This section contains the following topics:

Understanding System Variables

Introduction

This section describes how System Variables are implemented for the controller. These variables have the following attributes:

- System Variables allow you to access general system information, perform system diagnostics, and command simple actions.
- System Variables are structured variables conforming to IEC 61131-3 definitions and naming conventions. You can access the System Variables using 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`).
- System Variables are automatically declared as global variables. They have system-wide scope and must be handled with care because they can be accessed by any Program Organization Unit (POU) in any task.

System Variables Naming Convention

The System Variables are identified by:

- a structure name which represents the category of System Variable (for example, `PLC_R` represents a structure name of read only variables used for the controller diagnosis).
- a set of component names which identifies the purpose of the variable (for example, `i_wVendorID` represents the controller Vendor ID).

You can access the 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.

System Variables Location

One type of system variable that is defined for use when programming the controller is unlocated variables.

These unlocated variables can only be accessed via login or sharing the symbol in **Symbol Configuration** to HMI.

Using System Variables

Introduction

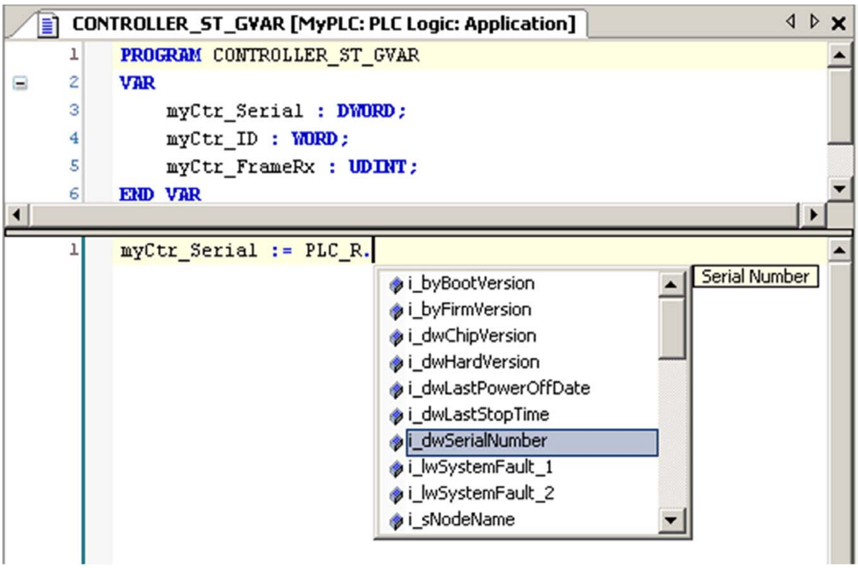
This section describes the steps required to program and to use system variables in SoMachine.

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

The following table describes the parameters of the PLC_R System Variable (PLC_R_STRUCT type):

Var Name	Type	Comment
i_wVendorID	WORD	Controller Vendor ID. 101A hex = Schneider Electric
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).
i_dwSerialNumber	DWORD	Controller serial number (returns last five digits).
i_byFirmVersion[0..3]	ARRAY[0..3] OF BYTE	Controller Firmware Version [aa.bb.cc.dd]: ● i_byFirmVersion[0] = aa ● ... ● i_byFirmVersion[3] = dd
i_byBootVersion[0..3]	ARRAY[0..3] OF BYTE	Controller Boot Version [aa.bb.cc.dd]: ● i_byBootVersion[0] = aa ● ... ● i_byBootVersion[3] = dd
i_dwHardVersion	DWORD	Controller Hardware Version.
i_dwChipVersion	DWORD	Controller Processor Version.
i_wStatus	PLC_R_STATUS *(see page 40)*	State of the controller.
i_wBootProjectStatus	PLC_R_BOOT_ PROJECT_STATUS *(see page 38)*	Returns information about the boot application stored in FLASH memory.
i_wLastStopCause	PLC_R_STOP_ CAUSE *(see page 41)*	Cause of the last transition from RUN to another state.
i_wLastApplicationError	PLC_R_APPLICATION_ ERROR *(see page 37)*	Cause of the last controller exception.
i_lwSystemFault_1	LWORD	Not used. always returns FFFF FFFF FFFF FFFF hex.
i_lwSystemFault_2	LWORD	Not used.

Var Name	Type	Comment
i_wIOStatus1	PLC_R_IO_STATUS *(see page 39)*	Embedded I/O status.
i_wIOStatus2	PLC_R_IO_STATUS	Reserved.
i_wBatteryStatus	WORD	Charge remaining in the battery. This system variable can report the following significant values: • 0064 hex = 100% = 3 V • 0032 hex = 50% = 2.5 V • 0000 hex = 0% = 2 V **NOTE:** HMI controller display screen indicates also that the battery level is low at 50% or lower.
i_dwAppliSignature1	DWORD	First DWORD of four DWORD signature (16 bytes total). The application signature is automatically generated by the software during build.
i_dwAppliSignature2	DWORD	Second DWORD of four DWORD signature (16 bytes total). The application signature is automatically generated by the software during build.
i_dwAppliSignature3	DWORD	Third DWORD of four DWORD signature (16 bytes total). The application signature is automatically generated by the software during build.
i_dwAppliSignature4	DWORD	Fourth DWORD of four DWORD signature (16 bytes total). The application signature is automatically generated by the software during build.

PLC_W: Controller Read / Write System Variables

Variable Structure

The following table describes the parameters of the PLC_W System Variable (PLC_W_STRUCT type):

Var Name	Type	Comment
q_uiOpenPLCControl	UINT	You are able to trigger either a **STOP**, **RUN**, **Reset Cold**, or **Reset Warm** command by using this variable.
q_wPLCControl	PLC_W_COMMAND *(see page 42)*	This variable contains the value representing a command for a state change desired by the user. Controller **STOP**, **RUN**, **Reset Cold**, and **Reset Warm** are enumerated in the Data Type PLC_W_COMMAND. This desired state change is triggered by changing the value of PLC_W.q_uiOpenPLCControl from 0 to 6699.

Section 1.3
SERIAL_W Structures

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

Introduction

SERIAL_W is a SERIAL_W_STRUCT type system variable. An element forces the SERIAL_R **System Variables** for the corresponding Serial Line to be reset.

For HMI SCU Serial_W[0] refers to the COM1.

Variable Structure

The following table describes the parameters of the SERIAL_W[0] System Variable:

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

NOTE: There is one communication port defined for the HMI SCU.

Chapter 2
HMI SCU System Functions

Section 2.1
HMI SCU Read Functions

Overview

This section describes the read functions included in the HMI SCU PLCSystem library.

What Is in This Section?

This section contains the following topics:

GetBatteryLevel: Returns Remaining Power Charge of the Battery

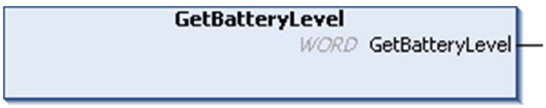
Function Description

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

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

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

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 51)*.

I/O Variables Description

The table describes the output variable:

Output	Type	Comment
`GetBatteryLevel`	`WORD`	Percentage of charge remaining in the battery. Range: 0...100: ● 100%= 3 V ● 50% = 2.5 V ● 0%= 2 V **NOTE:** On the HMI controller a `Battery Level is low` message will be displayed if the voltage ≤ 2.5 Volts.

`GetLocalAIOStatus`: Returns the Embedded Analog I/O Status

Function Description

This function returns the status of the local input and outputs.

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 51)*.

I/O Variable Description

The table describes the output variable:

Bit	Description
0	Analog input 0 with configuration detected error.
1	Analog input 0 with current out of range.
2	Analog input 0 with invalid data.
3	Analog input 0 with broken wire (only available when input range is 4-20 mA).
4	Analog input 1 with configuration detected error.
5	Analog input 1 with current out of range.
6	Analog input 1 with invalid data.
7	Analog input 1 with broken wire (only available when input range is 4-20 mA).
8	Analog output 0 with configuration detected error.
9	Analog output 0 with voltage or current out of range.
10	Analog output 0 with invalid data.
11	Analog output 0 with detected error.
12	Analog output 1 with configuration detected error.
13	Analog output 1 with voltage or current out of range.
14	Analog output 1 with invalid data.
15	Analog output 1 with detected error.

`GetLocalIOStatus`: Returns the Embedded I/O Status

Function Description

This function returns the embedded I/O status.

NOTE: This information is also available through the System Variable `PLC_R.i_wLocalIO-Status` *(see page 18)*.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation (see page 51)*.

I/O Variables Description

The table describes the input parameter:

Input	Type	Comment
`Mode`	`LOCAL_IO_GET_STATUS` *(see page 47)*	Parameter of the function: currently only LOCAL_IO_GET_GEN_STATUS (00 hex) *(see page 47)* is available.

The table describes the output variable:

Output	Type	Comment
`GetLocalIOStatus`	`LOCAL_IO_GEN_STATUS` *(see page 48)*	Status of the embedded I/O.

Example 1

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

```
VAR
     MyLocalIOStatus : LOCAL_IO_GEN_STATUS;
END_VAR

MyLocalIOStatus := GetLocalIOStatus(LOCAL_IO_GET_GEN_STATUS);
```

Example 2

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

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

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

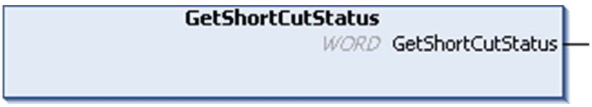
GetShortCutStatus: Returns the Short-Circuit Status on Embedded Outputs

Function Description

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

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

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 51)*.

I/O Variable Description

The table describes the output variable:

Parameter	Type	Comment
GetShortCutStatus	WORD	See bit field description below.

The table describes the bit field for the controller:

Bit	Description
0	TRUE = short-circuit on outputs (Q0 and Q1).

`GetTempStatus`: Function Returns the Temperature Input Status

Function Description

This function returns the status of the temperature inputs.

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 51)*.

I/O Variable Description

Parameter	Type	Comment
`GetTempStatus`	`WORD`	See bit field description below.

The table describes the input variable:

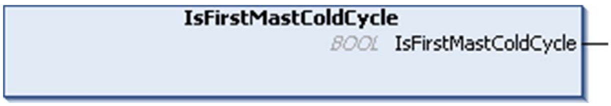
Bit	Comment
0	Temperature input 0 with configuration error detected.
1	Temperature input 1 with configuration error detected.
2	Temperature input 0 with over range error detected.
3	Temperature input 1 with over range error detected.
4	Temperature input 0 with invalid data.
5	Temperature input 1 with invalid data.
6	Temperature input 0 with broken wire detected.
7	Temperature input 1 with broken wire detected.
8...15	Reserved.

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

Function Description

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

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 51)*.

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.

`IsFirstMastCycle`: Indicates if Cycle is the First MAST Cycle

Function Description

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

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 51)*.

I/O Variable Description

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

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

Function Description

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

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 51)*.

I/O Variable Description

The table describes the output variable:

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

Chapter 3
HMI SCU PLCSystem Library Data Types

Overview

This chapter describes the **DataTypes** of the HMI SCU PLCSystem Library.

There are two kinds of **Data Types** available:

- **System Variable Data types** are used by the **System Variables** *(see page 18)* of the HMI SCU PLCSystem Library (PLC_R, PLC_W,...).
- **System Function Data Types** are used by the read/write **System Functions** *(see page 23)* of the HMI SCU PLCSystem Library.

What Is in This Chapter?

This chapter contains the following sections:

Section 3.1
PLC_R/W System Variables Data Types

Overview

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

What Is in This Section?

This section contains the following topics:

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
PLC_R_APP_ERR_UNKNOWN	FFFF hex	Undefined error.
PLC_R_APP_ERR_NOEXCEPTION	0000 hex	No error has been detected.
PLC_R_APP_ERR_WATCHDOG	0010 hex	Application watchdog expired.
PLC_R_APP_ERR_HARDWAREWATCHDOG	0011 hex	Hardware watchdog expired.
PLC_R_APP_ERR_IO_CONFIG_ERROR	0012 hex	Incorrect I/O configuration parameters detected.
PLC_R_APP_ERR_UNRESOLVED_EXTREFS	0018 hex	Undefined functions detected.
PLC_R_APP_ERR_IEC_TASK_CONFIG_ERROR	0025 hex	Incorrect Task configuration parameters detected.
PLC_R_APP_ERR_ILLEGAL_INSTRUCTION	0050 hex	Undefined instruction detected.
PLC_R_APP_ERR_ACCESS_VIOLATION	0051 hex	Attempted access to reserved memory area.
PLC_R_APP_ERR_DIVIDE_BY_ZERO	0102 hex	Integer division by zero detected.
PLC_R_APP_ERR_DIVIDE_REAL_BY_ZERO	0152 hex	Real division by zero detected.

PLC_R_BOOT_PROJECT_STATUS: Boot Project Status Codes

Enumerated Type Description

The PLC_R_BOOT_PROJECT_STATUS enumeration data type contains the following values:

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

PLC_R_IO_STATUS: I/O Status Codes

Enumerated Type Description

The PLC_R_IO_STATUS enumeration data type contains the following values:

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

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
PLC_R_STOP_REASON_UNKNOWN	00 hex	Initial value or stop cause is undefined.
PLC_R_STOP_REASON_HW_WATCHDOG	01 hex	Stopped after hardware (system hardware or software task) watchdog timeout.
PLC_R_STOP_REASON_RESET	02 hex	Stopped after reset.
PLC_R_STOP_REASON_EXCEPTION	03 hex	Stopped after exception.
PLC_R_STOP_REASON_USER	04 hex	Stopped after a user request.
PLC_R_STOP_REASON_IECPROGRAM	05 hex	Stopped after a program command request.
PLC_R_STOP_REASON_DELETE	06 hex	Stopped after a remove application command.
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 `(PLC_W.q_wPLCControl:=PLC_W_COMMAND.PLC_W_STOP;).`
PLC_R_STOP_FROM_INPUT	0B hex	Stop required by a controller input.

For more information for reasons the controller has stopped, refer to the Controller State Description *(see Magelis SCU, HMI Controller, Programming Guide)*.

PLC_W_COMMAND: Control Command Codes

Enumerated Type Description

The PLC_W_COMMAND enumeration data type contains the following values:

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

Section 3.2
System Function Data Types

Overview

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

What Is in This Section?

This section contains the following topics:

FIRMWARE_VERSION: GetFirmwareVersion Function Output Type

Structure Description

The data structure contains the following variables:

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

BOOT_PROJECT_STATUS: GetBootProjectStatus Function Output Codes

Enumerated Type Description

The enumeration data type contains the following values:

Enumerator	Value	Description
NO_BOOT_PROJECT	0000 hex	Not used.
BOOT_PROJECT_CREATION_IN_PROGRESS	0001 hex	Boot project is being created.
DIFFERENT_BOOT_PROJECT	0002 hex	Boot project in Flash is different from the project loaded in RAM or a boot project does not exist.
VALID_BOOT_PROJECT	FFFF hex	Boot project in Flash is the same as the project loaded in RAM.

STOP_WHY: `GetLastStopCause` Function Output Codes

Enumerated Type Description

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

Enumerator	Value	Description
STOP_REASON_UNKNOWN	00 hex	Initial value or stop cause is undefined.
STOP_REASON_HW_WATCHDOG	01 hex	Stopped after software watchdog timeout.
STOP_REASON_RESET	02 hex	Stopped after reset.
STOP_REASON_EXCEPTION	03 hex	Stopped after exception.
STOP_REASON_USER	04 hex	Stopped after a user request.
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;).
STOP_REASON_DELETE	06 hex	Stopped after a remove application command.
STOP_REASON_DEBUGGING	07 hex	Stopped after entering debug mode.
STOP_FROM_NETWORK_REQUEST	0A hex	Stopped after a request from the network (PLC_W command).
STOP_FROM_INPUT	0B hex	Stop required by a controller input.

LOCAL_IO_GET_STATUS: GetLocalIOStatus Function Parameter Codes

Enumerated Type Description

The LOCAL_IO_GET_STATUS enumeration data type contains the following value:

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

LOCAL_IO_GEN_STATUS: GetLocalIOStatus Function Output Codes

Enumerated Type Description

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

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

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.

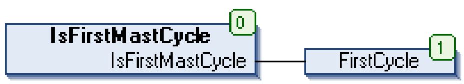
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

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 SoMachine POU IL Editor

IL example of a function without input parameter:
`IsFirstMastCycle`
	`PROGRAM MyProgram_IL`
	`VAR`
	`    FirstCycle: BOOL;`
	`END_VAR`
	
	`IsFirstMastCycle`
	`ST                FirstCycle`

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

Using a Function Block in IL Language

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

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function block requires, including the instance name.
3	Function Blocks are called using a `CAL` instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the `CAL` instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values to inputs are set by "`:=`". ● Values to outputs are set by "`=>`".
4	In the `CAL` right-side field, replace `???` with the instance name.
5	Replace other `???` with an appropriate variable or immediate value.

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

Function Block	Graphical Representation
`TON`	

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

<table>
<tr><th>Function Block</th><th>Representation in SoMachine POU IL Editor</th></tr>
<tr><td>TON</td><td>

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

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

</td></tr>
</table>

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 SoMachine 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:

<table>
<tr><th>Function Block</th><th>Representation in SoMachine POU ST Editor</th></tr>
<tr><td>TON</td><td>

```
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);
```

</td></tr>
</table>

Glossary

A

application

A program including configuration data, symbols, and documentation.

B

byte

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

C

CFC

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

configuration

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

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).

E

expansion bus

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

F

FB

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

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.

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 SoMachine project.

I

I/O

(*input/output*)

IL

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

INT

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

L

LD

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

M

MAST

A processor task that is run through its programming software. The MAST task has 2 sections:

- **IN:** Inputs are copied to the IN section before execution of the MAST task.
- **OUT:** Outputs are copied to the OUT section after execution of the MAST task.

P

POU

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

program

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

S

ST

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

V

variable

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

Index

S