Altivar ATV IMC Drive Controller

System Functions and Variables
ATV-IMC PLCSystem Library Guide

12/2015

EIO0000000596.05

www.schneider-electric.com

Schneider Electric

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

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

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

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

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

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

© 2015 Schneider Electric. All rights reserved.

Table of Contents

Safety Information

Important Information

NOTICE

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

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

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

⚠ DANGER

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

⚠ WARNING

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

⚠ CAUTION

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

NOTICE

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

PLEASE NOTE

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

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

About the Book

At a Glance

Document Scope

This document acquaints you with the system functions and variables offered within the Altivar ATV IMC Drive Controller. The ATV IMC PLCSystem library contains functions and variables to get information from and to send commands to the Altivar ATV IMC Drive Controller system.

This documentation describes the data types functions and variables of the ATV IMC PLCSystem library.

The following basic knowledge is required:

- basic information on functionality, structure and configuration of the ATV IMC
- programming in the FBD, LD, ST, IL, SFC or CFC language
- System Variables (Global Variables)

Validity Note

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

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.

Standard	Description
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2004/108/EC	Electromagnetic Compatibility Directive
2006/95/EC	Low Voltage Directive

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

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

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

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

Chapter 1
ATV IMC System Variables

Overview

This chapter:

- gives an introduction to the system variables *(see page 12)*
- describes the system variables *(see page 18)* included with the ATV IMC 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 Altivar ATV IMC Drive 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 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 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

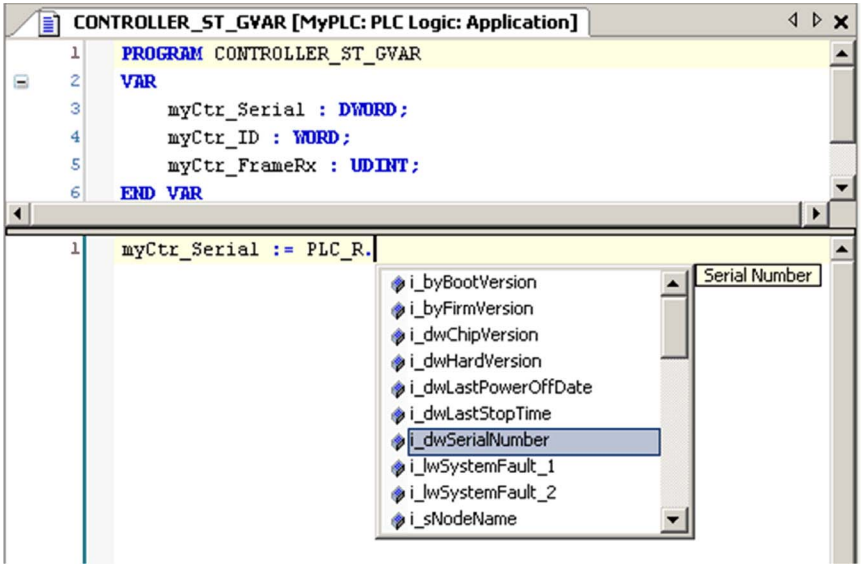
This section describes the steps required to program and to use system variables in SoMachine.

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_dwHardwareID	DWORD	Controller Coprocessor Version.
60012	i_wStatus	PLC_R_STATUS *(see page 48)*	State of the controller.
60013	i_wBootProjectStatus	PLC_R_BOOT_PROJECT_STATUS *(see page 46)*	Returns information about the boot application stored in FLASH memory.
60014	i_wLastStopCause	PLC_R_STOP_CAUSE *(see page 49)*	Cause of the last transition from RUN to another state.
60015	i_wLastApplicationError	PLC_R_APPLICATION_ERROR *(see page 43)*	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 = error detected on ATV-IMC internal link ● bit 1 = Ethernet link not connected ● bit 2 = USB link not connected ● bit 3 = CANopen link not running ● bit 4 = Modbus/TCP time-out ● bit 5 = Duplicate IP address detected ● bit 6 = Overload detected on Ethernet network ● bit 7 = error detected on Ethernet hardware ● bit 8 = error detected on non-volatile memory ● bit 9 = CAN communication messaging error detected ● bit 10 = error detected on ATV-IMC object dictionary ● bit 11 = System watchdog error detected ● bit 12 = Internal error detected ● bit 13 = Logical output error detected (over temperature) ● bit 14 = Logical output 24 V power supply inoperative ● bit 15-63: Not used **NOTE:** Bit 11 and bit 12 can be reset using the function ResetInternalErrorDiag *(see page 37)*.
60020	i_lwSystemFault_2	LWORD	Not used.
60024	i_wIOStatus1	PLC_R_IO_STATUS *(see page 47)*	Embedded I/O status.
60025	i_wIOStatus2	PLC_R_IO_STATUS *(see page 47)*	not used (always FFFF hex).
60026	i_wBatteryStatus	PLC_R_BATTERY_ STATUS *(see page 45)*	Real Time Clock battery status.
60028	i_dwAppliSignature1	DWORD	First 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
60030	i_dwAppliSignature2	DWORD	Second DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
60032	i_dwAppliSignature3	DWORD	Third DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
60034	i_dwAppliSignature4	DWORD	Fourth DWORD of 4 DWORD signature (16 bytes total). The application signature is generated by the software during build.
(1) Not accessible through the application.			

n/a	i_sVendorName	STRING(31)	Name of the vendor: “Schneider Electric”.
n/a	i_sProductRef	STRING(31)	Reference of the Controller.

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
62000	q_uiOpenPLCControl	UINT	When Value passes from 0 to 6699, the command previously written in the following PLC_W.q_wPLCControl is executed.
62001	q_wPLCControl	PLC_W_COMMAND *(see page 50)*	Controller RUN / STOP command executed when the system variable PLC_R.q_uiOpenPLCControl value passes from 0 to 6699.

Section 1.3
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.
60067	i_wIpMode	ETH_R_IP_MODE *(see page 52)*	Method used to obtain an IP address.
60068	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.
60070	i_udiOpenTcpConnections	UDINT	Number of open TCP connections.
60073	i_udiFramesTransmittedOK	UDINT	Number of frames successfully transmitted. 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.			

%MW	Var Name	Type	Comment
60075	i_udiFramedReceivedOK	UDINT	Number of frames successfully received. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
60077	i_udiTransmitBufferErrors	UDINT	Numbers of frames transmitted with detected errors. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
60079	i_udiReceiveBufferErrors	UDINT	Numbers of frames received with detected errors. Reset at Power ON or with reset command ETH_W.q_wResetCounter.
60081	i_wPortALinkStatus	ETH_R_PORT_LINK_STATUS *(see page 55)*	Link of the Ethernet Port (0 = No Link, 1 = Link connected to another Ethernet device).
60082	i_wPortASpeed	ETH_R_PORT_SPEED *(see page 56)*	Ethernet Port network speed (10Mb/s or 100Mb/s).
60083	i_wPortADuplexStatus	ETH_R_PORT_DUPLEX_STATUS *(see page 54)*	Ethernet Port duplex status (0 = Half or 1 = Full duplex).
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
62066	q_wResetCounter	WORD	Transition from 0 to 1 resets all ETH_R counters. To reset again, it is necessary to write this register to 0 before another transition from 0 to 1 can take place.

Chapter 2
ATV IMC System Functions

Overview

This chapter describes the system functions included in the ATV IMC PLCSystem library.

What Is in This Chapter?

This chapter contains the following sections:

Section 2.1
ATV IMC Read Functions

Overview

This section describes the read functions included in the ATV IMC PLCSystem library.

What Is in This Section?

This section contains the following topics:

GetEventsNumber: Returns the Number of External Events Detected

Function Description

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

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the *Function and Function Block Representation* *(see page 65)* chapter.

I/O Variable Description

The following table describes the output variable:

Output	Type	Comment
GetEventsNumber	UINT	The value representing the number of events that have occurred since the last cold start. Reset to 0 with a call of the function ResetEventsNumber *(see page 36)*.

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

Function Description

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

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the *Function and Function Block Representation* *(see page 65)* chapter.

I/O Variable Description

The following table describes the output variable:

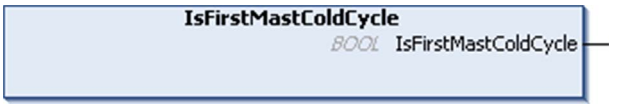
Output	Type	Comment
`GetLastStopTime`	`DWORD`	The time of the last detected STOP in seconds beginning with January 1, 1970 at 00:00.

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 *Function and Function Block Representation* *(see page 65)* chapter.

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 32)*.

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 *Function and Function Block Representation* *(see page 65)* chapter.

I/O Variable Description

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

Example

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

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

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

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

IF (MyIsFirstMastWarmCycle) THEN

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

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

END_IF;
```

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

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

Function Description

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

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the *Function and Function Block Representation* *(see page 65)* chapter.

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 32)*.

Section 2.2
ATV IMC Write Functions

Overview

This section describes the write functions included in the ATV IMC PLCSystem library.

What Is in This Section?

This section contains the following topics:

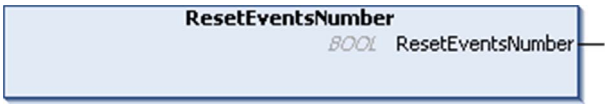
ResetEventsNumber: Resets Events Number

Function Description

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

NOTE: The number of events detected on inputs or HSC threshold detection is returned by the function `GetEventsNumber` *(see page 29)*.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the *Function and Function Block Representation* *(see page 65)* chapter.

I/O Variable Description

The following table describes the output variable:

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

ResetInternalErrorDiag: Resets Flags Raised upon a System Watchdog or a Detected Internal Error

Function Description

This function resets flags raised on a system watchdog or on a detected internal error.

NOTE: A system watchdog and a detected internal error are flagged respectively by bit 11 and bit 12 of `i_lwSystemFault_1` register in `PLC_R` *(see page 18)* System Variable.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the *Function and Function Block Representation (see page 65)* chapter.

I/O Variable Description

The following table describes the input variable:

Input	Type	Comment
`error`	`PLC_ERROR_TYPE` *(see page 62)*	Detected error to reset

Example

```
VAR
     hw_watchdog_error : PLC_ERROR_TYPE := _ERR_IMC_WATCHDOG
     internal_error : PLC_ERROR_TYPE := _ERR_IMC_INT_ERROR
END_VAR

ResetInternalErrorDiag(hw_watchdog_error);
ResetInternalErrorDiag(internal_error);
```

SetLEDBehaviour: Determines the Behavior of a LED

Function Description

This function controls the application LED USER.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the *Function and Function Block Representation (see page 65)* chapter.

I/O Variables Description

The following table describes the input parameters:

Inputs	Type	Comment
LedId	LED_ID *(see page 61)*	ID of the application LED.
LedColor	LED_COLOR *(see page 60)*	Color of the application LED.
LedBhv	LED_BHV *(see page 58)*	Mode of the application LED.

The following table describes the output variable:

Output	Type	Comment
SetLEDBehaviour	LED_BHV_ERROR *(see page 59)*	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 USER 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

Chapter 3
ATV IMC PLCSystem Library Data Types

Overview

This chapter describes the data types of the ATV IMC PLCSystem Library.

There are 2 kinds of data types available:

- System variable data types are used by the system variables *(see page 11)* of the ATV IMC PLCSystem Library (PLC_R, PLC_W,...).
- System function data types are used by the read/write system functions *(see page 27)* of the ATV IMC 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	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. Run → Build → Clean All** **2.** Export/Import your application. **3.** Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_UNRESOLVED_EXTREFS	0018 hex	Undefined functions detected.	Delete the unresolved functions from the application.
PLC_R_APP_ERR_IEC_TASK_CONFIG_ERROR	0025 hex	Incorrect Task configuration parameters detected.	Your application might be corrupted. To resolve this issue, use one of the methods: **1. Run → Build → Clean All** **2.** Export/Import your application. **3.** Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_ILLEGAL_INSTRUCTION	0050 hex	Undefined instruction detected.	Debug your application to resolve the problem.
PLC_R_APP_ERR_ACCESS_VIOLATION	0051 hex	Attempted access to reserved memory area.	Debug your application to resolve the problem.
PLC_R_APP_ERR_RTSEXCPT_DIVIDEBYZERO	0102 hex	Integer division by zero detected.	Debug your application to resolve the problem.

Enumerator	Value	Comment	What to do
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_RTSEXCPT_FPU_DIVIDEBYZERO	0152 hex	Floating point division by zero detected.	Debug your application to resolve the problem.

PLC_R_BATTERY_STATUS: Battery Status Codes

Enumerated Type Description

The PLC_R_BATTERY_STATUS enumeration data type contains the following values:

Enumerator	Value	Description
BATTERY_OK	0000 hex	Battery is Ok and Real Time Clock (RTC) is configured.
BATTERY_NOT_CONFIGURED	0001 hex	Battery is Ok but RTC is not configured.
BATTERY_UNPLUGGED	0002 hex	Battery is unplugged or its change is needed.

PLC_R_BOOT_PROJECT_STATUS: Boot Project Status Codes

Enumerated Type Description

The PLC_R_BOOT_PROJECT_STATUS enumeration data type contains the following values:

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

PLC_R_IO_STATUS: I/O Status Codes

Enumerated Type Description

The PLC_R_IO_STATUS enumeration data type contains the following values:

Enumerator	Value	Comment
PLC_R_IO_OK	FFFF hex	Inputs/Outputs are operational.
PLC_R_IO_NO_INIT	0001 hex	Inputs/Outputs are not initialized.
PLC_R_IO_CONF_FAULT	0002 hex	Incorrect I/O configuration parameters detected.
PLC_R_IO_SHORTCUT_FAULT	0003 hex	Inputs/Outputs short-circuit detected.
PLC_R_IO_POWER_SUPPLY_FAULT	0004 hex	Inputs/Outputs power supply error detected.

PLC_R_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_FROM_STOP	00 hex	Controller stopped due to an application stop request	–
PLC_R_STOP_FROM_POWER_FAIL	01 hex	Controller stopped due to a power outage	–
PLC_R_STOP_FROM_NETWORK_REQUEST	02 hex	Stopped after a request from the network (USB key or PLC_W command).	–
PLC_R_HALT_FROM_APP_WATCHOG	03 hex	Controller stopped due to a task watchdog event	Check your application. See chapter . A reset is needed to enter Run mode.
PLC_R_HALT_FROM_CPU_OVERLOAD	04 hex	Controller stopped due to a CPU overload	Reduce the application workload by improving the application architecture. Increase the task cycle duration. Reduce event frequency.
PLC_R_HALT_FROM_INTERNAL_ERROR	05 hex	Controller stopped due to an internal detected error	Contact your local support.

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
ETH_R/W 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.

Section 3.3
System Function Data Types

Overview

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

What Is in This Section?

This section contains the following topics:

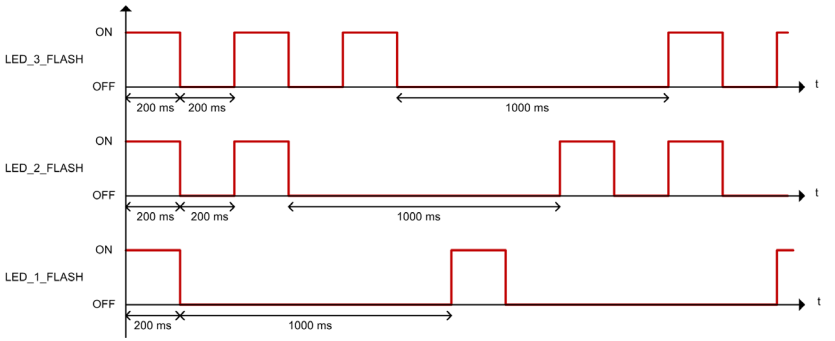
LED_BHV: SetLedBehaviour Function LedBhv Parameter Codes

Enumerated Type Description

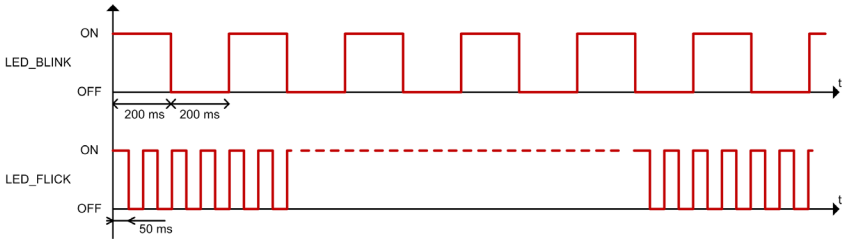
The LED_BHV enumeration data type contains the following values:

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.

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.

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 USER.

PLC_ERROR_TYPE: `ResetInternalErrorDiag` Function `error` Parameter Codes

Structure Description

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

Enumerator	Value	Description
`_ERR_IMC_WATCHDOG`	0B hex	System watchdog detected error
`_ERR_IMC_INT_ERROR`	0C hex	Other internal detected error

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.

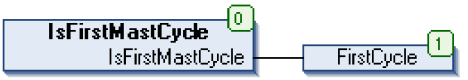
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

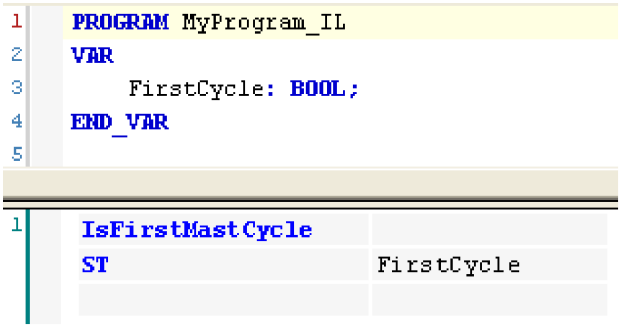
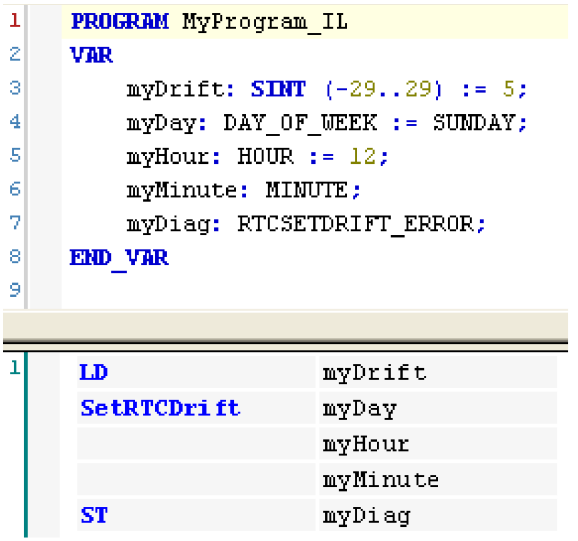
This procedure describes how to insert a function in IL language:

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`	
IL example of a function with input parameters: `SetRTCDrift`	

Using a Function Block in IL Language

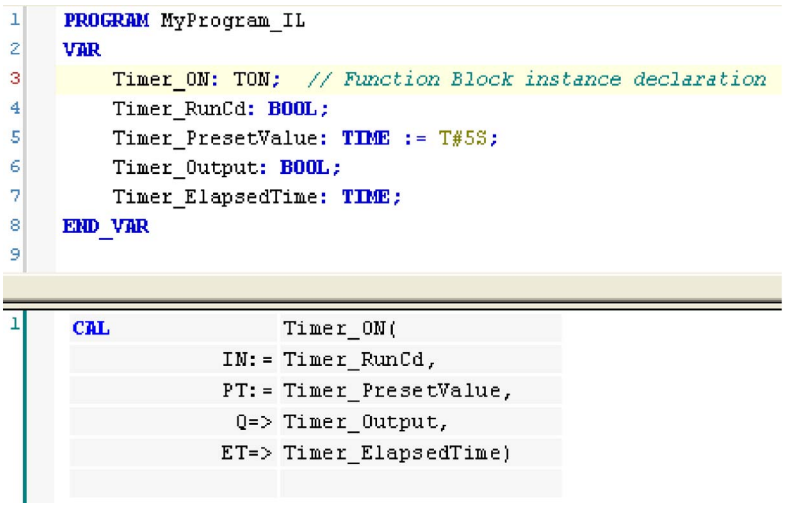
This procedure describes how to insert a function block in IL language:

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.
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 SoMachine POU IL Editor
TON	

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

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

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

General Information

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

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

Using a Function in ST Language

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

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see 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

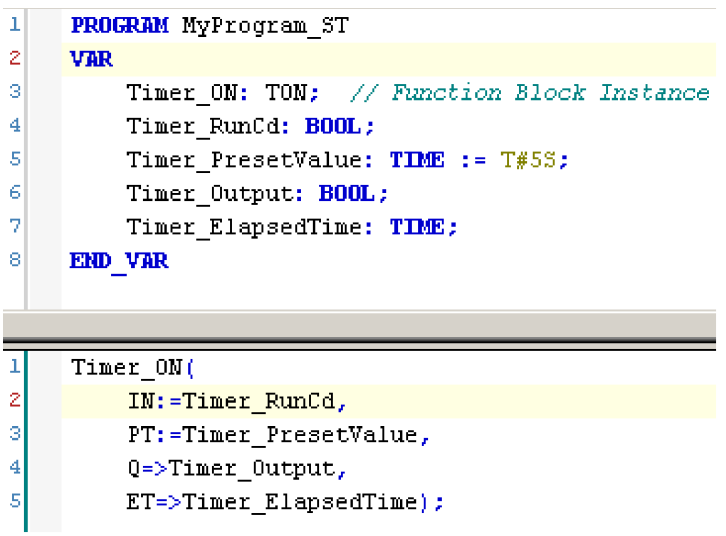
This procedure describes how to insert a function block in ST language:

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:

Function Block	Representation in SoMachine 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

CFC

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

configuration

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

controller

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

D

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.

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 block diagram

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

G

GVL

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

H

hex

(*hexadecimal*)

HSC

(*high-speed counter*)

I

I/O

(*input/output*)

ID

(*identifier/identification*)

IEC

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

IEC 61131-3

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

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).

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.

P

PLC

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

POU

(*program organization unit*) A variable declaration in source code and a corresponding instruction set. 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.

R

RTC

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

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.

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

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

P

R

S