Magelis XBTGC, XBTGT, XBTGK HMI Controller

System Functions and Variables
XBT PLCSystem Library Guide

11/2015

EIO0000000626.07

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 acquaints you with the system functions and variables offered within the XBTGC, XBTGK, and XBTGT Controllers. The XBT PLCSystem library contains functions and variables to get information from and to send commands to the XBT system.

This documentation also describes the functions and variables associated data types of the XBT PLCSystem library.

The following basic knowledge is required:

- basic information on functionality, structure and configuration of the XBTGC, XBTGT, and XBT GK HMI Controller
- 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
XBTGC, XBTGT, and XBTGK 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 XBT 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 XBTGC 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 definitions and naming conventions. They may be accessed 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`).
- 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 (e.g. `PLC_R` represents a structure name of read only variables used for controller diagnosis).
- a set of component names which identifies the purpose of the variable (e.g. `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. 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 between 0 and 59999 can be accessed directly; adresses greater than this are considered out of range by SoMachine and can only be accessed through the `structure_name.component_name` convention).

The unlocated variables:

- are not physically located in the %MW area
- are not accessible through any field bus or network requests
- are used in SoMachine programs according to the `structure_name.component_name` convention explained previously.

Using System Variables

Introduction

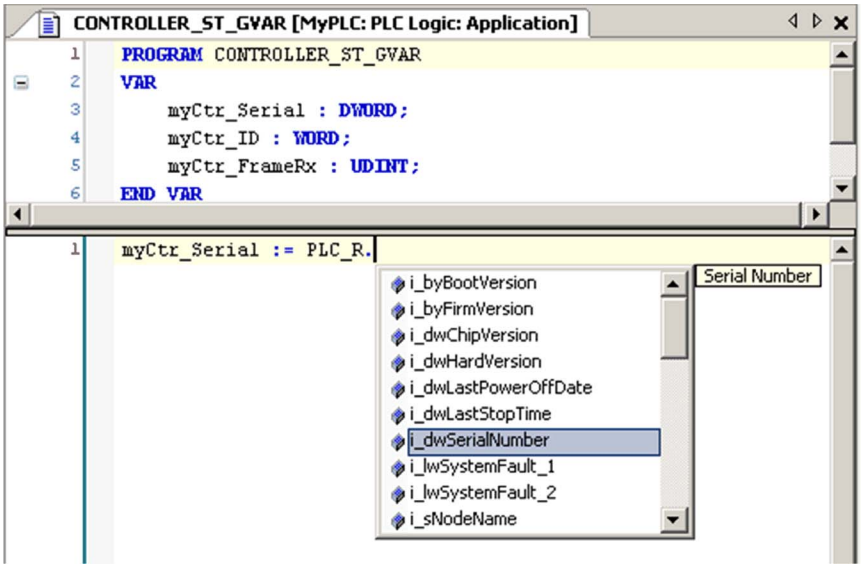
This section describes the steps required to program and to use system variables in SoMachine.

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_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_wStatus	PLC_R_STATUS *(see page 39)*	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 40)*	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 detected.
i_wIOStatus1	PLC_R_IO_STATUS *(see page 36)*	Embedded I/O status. **NOTE:** This is valid for XBT GC only.
i_wIOStatus2	PLC_R_IO_STATUS *(see page 36)*	TM2 I/O status. **NOTE:** This is valid for XBT GC only.
i_dwAppliSignature1	DWORD	1st DWORD of 4 DWORD signature (16 bytes total). The Application Signature is generated by the software during build.

Var Name	Type	Comment
i_dwAppliSignature2	DWORD	2nd DWORD of 4 DWORD signature (16 bytes total). The Application Signature is generated by the software during build.
i_dwAppliSignature3	DWORD	3rd DWORD of 4 DWORD signature (16 bytes total). The Application Signature is generated by the software during build.
i_dwAppliSignature4	DWORD	4th DWORD of 4 DWORD signature (16 bytes total). The Application Signature is generated by the software during build.

PLC_W: Controller Read / Write System Variables

Variable Structure

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

Var Name	Type	Comment
q_uiOpenPLCControl	UINT	When Value pass from 0 to 6699, The command previously written in the following PLC_W.q_wPLCControl is executed.
q_wPLCControl	PLC_W_COMMAND *(see page 41)*	Controller RUN / STOP command executed when the system variable PLC_R.q_uiOpenPLCControl value pass from 0 to 6699.

Section 1.3
SERIAL_R and SERIAL_W Structures

Overview

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

What Is in This Section?

This section contains the following topics:

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

Introduction

SERIAL_R is an array of 2 SERIAL_R_STRUCT type elements. Each element of the array returns diagnostic **System Variables** for the corresponding Serial Line:

- Serial_R[0] refers to the COM1 port (not relevant for XBTGC1100).
- Serial_R[1] refers to the COM2 port (not relevant for XBTGC series).

Variable Structure

The following table describes the parameters of the SERIAL_R[0..1] System Variable:

Var Name	Type	Comment
i_udiFramesTransmittedOK	UDINT	Number of frames successfully transmitted.
i_udiFramesReceivedOK	UDINT	Number of frames received without detected error.
i_udiRX_MessagesError	UDINT	Number of frames received with detected errors (checksum, parity).

NOTE:
The SERIAL_R Counters are reset on:

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

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

Introduction

SERIAL_W is an array of 2 SERIAL_W_STRUCT type elements. Each element of the array forces the SERIAL_R **System Variables** for the corresponding Serial Line to be reset:

- Serial_W[0] refers to the COM1 port (not relevant for XBTGC1100 series).
- Serial_W[1] refers to the COM2 port (not relevant for XBTGC series).

Variable Structure

The following table describes the parameters of the SERIAL_W[0..1] System Variable:

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

Chapter 2
XBTGC , XBTGT, and XBTGK System Functions

Section 2.1
XBTGC, XBTGT, and XBTGK Read Functions

Overview

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

What Is in This Section?

This section contains the following topics:

HMI_GetRightBusStatus: Returns the Status of the Expansion Bus

Function Description

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

NOTE: The `HMI_GetRightBusStatus` function is applicable for XBT GC HMI Controller only.

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 45)* chapter.

I/O Variables Description

The following table describes the input parameter:

Input	Type	Comment
`Mask`	`RIGHTBUS_GET_STATUS` *(see page 42)*	Defines the type of I/O expansion bus diagnostic returned by the function: bus configuration or one of the 3 possible expansion modules.

The following table describes the output variable:

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

I/O Expansion Bus Generic Diagnostic

The table below describes the bit field returned by the function `HMI_GetRightBusStatus` when the input parameter (`Mask`) is `RIGHT_BUS_GET_GEN_STATUS` *(see page 42)* for a configuration diagnostic of the expansion bus (0000 hex if no detected error):

Bit	Description
0	Reserved (always 0)
1	TRUE if the TM2 module 1 configuration mismatches with plugged module.
2	TRUE if the TM2 module 2 configuration mismatches with plugged module.
3	TRUE if the TM2 module 3 configuration mismatches with plugged module.
4..15	Reserved (always 0)

I/O Expansion Bus Modules Diagnostic

The tables below describe the bit field returned by the function `HMI_GetRightBusStatus` when the input parameter (`Mask`) is `RIGHTBUS_GET_STATUSx` *(see page 42)* (where x=1 to 3) for a detailed diagnostic of the expansion module "x".

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

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

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

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

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

When the associated module is one of the 4 or 8 analog input channels modules:
- TM2ARI8HT
- TM2AMI8HT
- TM2ARI8LT
- TM2ARI8LRJ
- TM2AMI4LT
- TM2AMM6HT

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

NOTE: When the targeted expansion module is a digital I/O module, the diagnostic returned is invalid (0000 hex).

Example

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

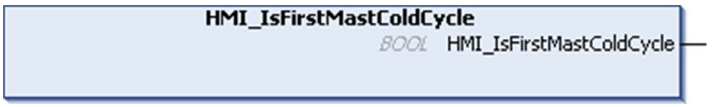
```
VAR
(*Modules 1 to 3 conf diag = MyRightBusStatus bits 1 to 3*)
MyRightBusStatus: WORD;
(*Modules 1 to 3 diag codes*)
    ModuleError:Array [1..3] of WORD;
END_VAR
(*Conf diagnostic on expansion bus*)
MyRightBusStatus:=HMI_GetRightBusStatus(RIGHTBUS_GET_GEN_STATUS);
IF MyRightBusStatus<>0 THEN
(*Conf mismatch detected => set an alarm, check bit values...*)END_IF;
(*Get modules diagnostic: detected error if diag <> 0*)
(*Limit the list to analog modules configured only*)
ModuleError[1]:=HMI_GetRightBusStatus(RIGHTBUS_GET_STATUS1);
ModuleError[2]:=HMI_GetRightBusStatus(RIGHTBUS_GET_STATUS2);
ModuleError[3]:=HMI_GetRightBusStatus(RIGHTBUS_GET_STATUS3);
```

HMI_IsFirstMastColdCycle: Indicates if Cycle is the First Mast Cold Start Cycle

Function Description

This function returns TRUE during the 1st 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 45)* chapter.

I/O Variable Description

The following table describes the output variable:

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

Example

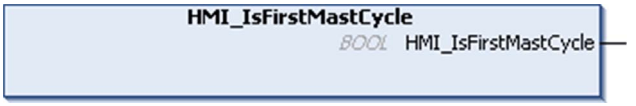
Refer to the function `HMI_IsFirstMastCycle` *(see page 32)*.

HMI_IsFirstMastCycle: Indicates if Cycle is the First Mast Cycle

Function Description

This function returns TRUE during the 1st 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 45)* chapter.

I/O Variable Description

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

Example

This example describes the three functions `HMI_IsFirstMastCycle`, `HMI_IsFirstMast-ColdCycle` and `HMI_IsFirstMastWarmCycle` used together:

NOTE: The functions must be used in MAST task otherwise initialization actions might be performed more than once or never (an additional task might be called several times or not called during 1 MAST task cycle).

```
IF HMI_IsFirstMastWarmCycle() THEN
(*This is the first Mast Cycle after a Warm Start: all variables are set
to their initialization values except the retained variables*)
(*=> initialize the needed variables in order your application to run as
expected in this case*)
END_IF;
IF HMI_IsFirstMastColdCycle() THEN
(*This is the first Mast Cycle after a Cold Start: all variables are set
to their initialization values including the Retains Variables*)
(*=> initialize the needed variables in order your application to run as
expected in this case*)
END_IF;
IF HMI_IsFirstMastCycle() THEN
(*This is the first Mast Cycle after a Start, ie after a Warm or Cold
Start as well as STOP/RUN commands*)
(*=> initialize the needed variables in order your application to run as
expected in this case*)
END_IF;]
```

HMI_IsFirstMastWarmCycle: Indicates if Cycle is the First Mast Warm Start Cycle

Function Description

This function returns TRUE during the 1st 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 45)* chapter.

I/O Variable Description

The following table describes the output variable:

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

Example

Refer to the function `HMI_IsFirstMastCycle` *(see page 32)*.

Chapter 3
XBT PLCSystem Library Data Types

Overview

This chapter describes the **DataTypes** of the XBT PLCSystem Library.

Two kinds of **Data Types** are available:

- **System Variable Data types** are used by the **System Variables** *(see page 11)* of the XBT PLCSystem Library (PLC_R, PLC_W,...).
- **System Function Data Types** are used by the read/write **System Functions** *(see page 25)* of the XBT PLCSystem Library.

What Is in This Chapter?

This chapter contains the following topics:

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	Invalid I/O configuration parameters detected.

PLC_R_APPLICATION_ERROR: Detected Application Error Status Codes

Enumerated Type Description

The PLC_R_APPLICATION_ERROR enumeration data type contains the following values:

Enumerator	Value	Comment	What to do
PLC_R_APP_ERR_UNKNOWN	FFFF hex	Undefined error detected.	Contact your local support.
PLC_R_APP_ERR_NOEXCEPTION	0000 hex	No error detected.	–
PLC_R_APP_ERR_WATCHDOG	0010 hex	Task watchdog expired.	Check your application. See chapter . A reset is needed to enter Run mode.
PLC_R_APP_ERR_HARDWAREWATCHDOG	0011 hex	System watchdog expired.	If the problem is reproducible, check for disconnected communication ports. If the problem persists, update the firmware. If the problem still persists, contact your local support.
PLC_R_APP_ERR_IO_CONFIG_ERROR	0012 hex	Incorrect I/O configuration parameters detected.	Your application might be corrupted. To resolve this issue, use one of the methods: **1. Build → Clean All** **2.** Export/Import your application. **3.** Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_UNRESOLVED_EXTREFS	0018 hex	Undefined functions detected.	Delete the unresolved functions from the application.
PLC_R_APP_ERR_IEC_TASK_CONFIG_ERROR	0025 hex	Incorrect Task configuration parameters detected.	Your application might be corrupted. To resolve this issue, use one of the methods: **1. Build → Clean All** **2.** Export/Import your application. **3.** Upgrade SoMachine to the latest version.
PLC_R_APP_ERR_ILLEGAL_INSTRUCTION	0050 hex	Undefined instruction detected.	Debug your application to resolve the problem.
PLC_R_APP_ERR_ACCESS_VIOLATION	0051 hex	Attempted access to reserved memory area.	Debug your application to resolve the problem.
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_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_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: RUN to Other State Transition Cause Codes

Enumerated Type Description

The PLC_R_STOP_CAUSE enumeration data type contains the following values:

Enumerator	Value	Comment	What to do
PLC_R_STOP_REASON_ UNKNOWN	0000 hex	Initial value or stop cause is undefined.	Contact your local support in case of undefined stop cause.
PLC_R_STOP_REASON_HW_ WATCHDOG	0001 hex	Stopped after hardware watchdog.	Contact your local support.
PLC_R_STOP_REASON_RESET	0002 hex	Stopped after reset.	See reset possibilities in chapter . A reset is needed to enter Run mode.
PLC_R_STOP_REASON_ EXCEPTION	0003 hex	Stopped after exception detected.	Check your application. See chapter .
PLC_R_STOP_REASON_USER	0004 hex	Stopped after a user request.	See chapter .
PLC_R_STOP_REASON_ IECPROGRAM	0005 hex	Stopped after a program command request (e.g.: control command with parameter PLC_W.q_wPLCControl:= PLC_W_COMMAND.PLC_W_ STOP;).	–
PLC_R_STOP_REASON_DELETE	0006 hex	Stopped after a remove application command.	See chapter .
PLC_R_STOP_REASON_ DEBUGGING	0007 hex	Stopped after entering debug mode.	–
PLC_R_STOP_FROM_NETWORK_ REQUEST	000A hex	Stopped after a request from the network (USB key or PLC_W command).	–
PLC_R_STOP_FROM_INPUT	000B hex	Stop required by a controller input.	–
PLC_R_STOP_REASON_RETAIN_ MISMATCH	000C hex	Sent to STOPPED state after a reboot and detection of a mismatch of remanent variables definition.	Retain variables were deleted because they are not referenced in the application. If the application sets retain variables to their intialization values, the is available.
PLC_R_STOP_REASON_BOOT_ APPLI_MISMATCH	000D hex	Sent to STOPPED state after a reboot and detection of a mismatch of boot application.	Create a valid boot application.
PLC_R_STOP_REASON_ POWERFAIL	000E hex	The controller has been stopped due to power interruption.	Check your power supply.

PLC_W_COMMAND: Control Command Codes

Enumerated Type Description

The PLC_W_COMMAND enumeration data type contains the following values:

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

RIGHTBUS_GET_STATUS: HMI_GetRightBusStatus Function Parameter Codes

Enumerated Type Descriptions

The enumeration data type contains the following values:

Enumerator	Value	Description
RIGHTBUS_GET_GEN_STATUS	00 hex	Parameter for an expansion bus configuration diagnostic
RIGHTBUS_GET_STATUS1	01 hex	Parameter for expansion bus Module 1 diagnostic
RIGHTBUS_GET_STATUS2	02 hex	Parameter for expansion bus Module 2 diagnostic
RIGHTBUS_GET_STATUS3	03 hex	Parameter for expansion bus Module 3 diagnostic

NOTE: For more information on using the RIGHTBUS_GET_STATUS parameter type, refer to the function HMI_GetRightBusStatus *(see page 27)*.

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.

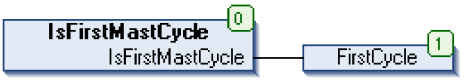
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

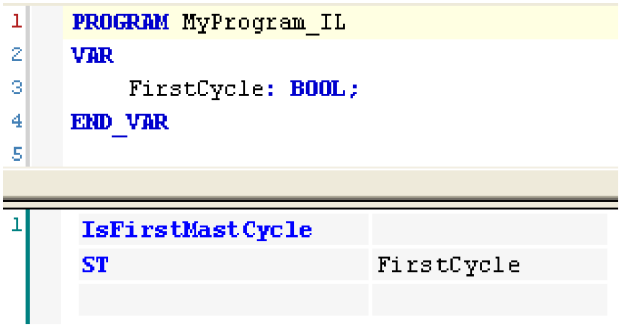
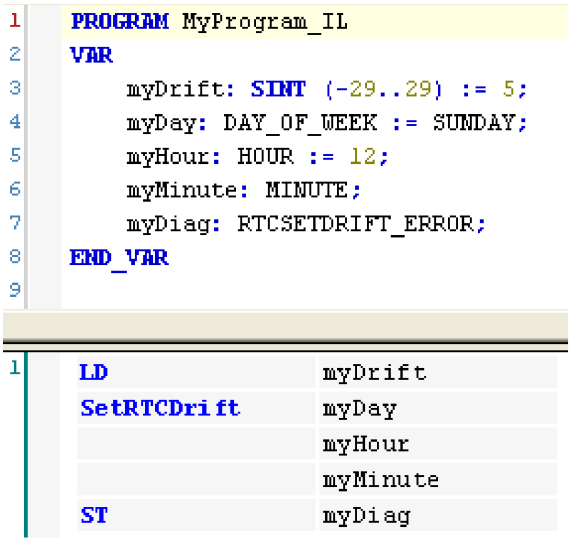
This procedure describes how to insert a function in IL language:

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

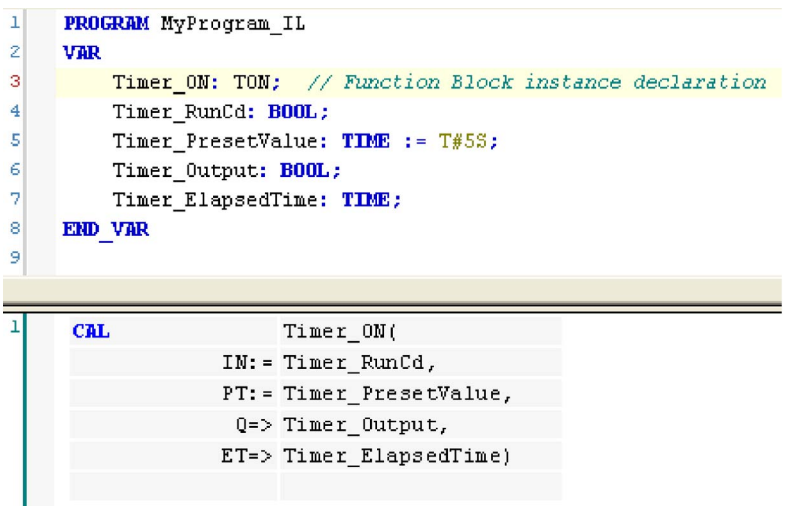
This procedure describes how to insert a function block in IL language:

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

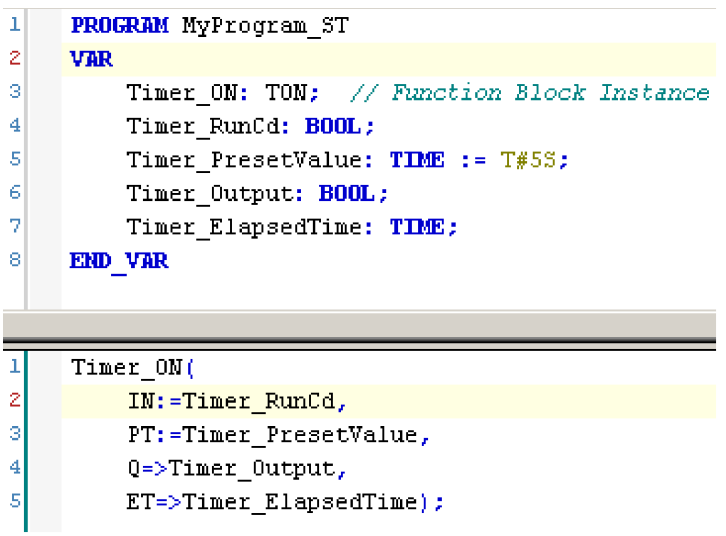
This procedure describes how to insert a function block in ST language:

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

analog input

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

application

A program including configuration data, symbols, and documentation.

ARRAY

The systematic arrangement of data objects of a single type in the form of a table defined in logic controller memory. The syntax is as follows: `ARRAY [<dimension>] OF <Type>`

Example 1: `ARRAY [1..2] OF BOOL` is a 1-dimensional table with 2 elements of type `BOOL`.

Example 2: `ARRAY [1..10, 1..20] OF INT` is a 2-dimensional table with 10 x 20 elements of type `INT`.

B

Boot application

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

byte

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

C

CFC

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

configuration

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

controller

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

D

digital I/O

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

DWORD

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

E

element

The short name of the ARRAY element.

expansion bus

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

F

FB

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

firmware

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

flash memory

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

function

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

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

function block diagram

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

G

GVL

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

H

hex

(*hexadecimal*)

HMI

(*human machine interface*) An operator interface (usually graphical) for human control over industrial equipment.

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.

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).

located variable

Refer to (*unlocated variable*).

M

MAST

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

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

Modbus

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

N

network

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

P

PLC

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

POU

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

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.

T

task

A group of sections and subroutines, executed cyclically or periodically for the MAST task or periodically for the FAST task.

A task possesses a level of priority and is linked to inputs and outputs of the controller. These I/O are refreshed in relation to the task.

A controller can have several tasks.

U

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.

X

XBT

Any Magelis XBT graphic panel.

Index