# TVDA Device Module Library

## Function Template Library Guide

06/2017

EIO0000001665.06

www.schneider-electric.com

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.

## BEFORE YOU BEGIN

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

### ⚠ WARNING

UNGUARDED EQUIPMENT

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

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

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only you, the user, machine builder or system integrator can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine and, therefore, can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, you should refer to the applicable local and national standards and regulations. The National Safety Council's Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

**NOTE:** Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

## START-UP AND TEST

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check be made and that enough time is allowed to perform complete and satisfactory testing.

**⚠ WARNING**

**EQUIPMENT OPERATION HAZARD**

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

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

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

**Software testing must be done in both simulated and real environments.**

Verify that the completed system is free from all short circuits and temporary grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:

- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

## OPERATION AND ADJUSTMENTS

The following precautions are from the NEMA Standards Publication ICS 7.1-1995 (English version prevails):

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.
- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer's instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer's instructions and the machinery used with the electrical equipment.
- Only those operational adjustments actually required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

# About the Book

## At a Glance

### Document Scope

This document describes the function templates of the TVDA Device Module Library.

### Validity Note

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

### Related Documents

| Title of Documentation | Reference Number |
|---|---|
| SoMachine EnergyEfficiencyToolbox Library Guide | EIO0000001157 (ENG) |
| SoMachine Industrial EtherNet, User Guide | EIO0000002215 |
| SoMachine Machine Energy Dashboard Library Guide | EIO0000001163 (ENG) |
| SoMachine Modbus and ASCII Read/Write Functions PLCCommunication Library Guide | EIO0000000361 (ENG) |
| SoMachine ModbusEnergyEfficiencyToolbox Library Guide | EIO0000001224 (ENG) |
| SoMachine PLCCommunication Library Guide | EIO0000000361 (ENG) |
| SoMachine, Programming Guide | EIO0000000067 (ENG) |
| SoMachine TeSys Motor Starters Functions TeSys Library Guide | EIO0000000657 (ENG) |
| Altivar Library Function Blocks Software Manual | 0198441113880 (ENG) |
| Lexium Library Function Blocks Software Manual | 0198441113892 (ENG) |
| LXM28 Library Function Blocks Software Manual | 0198441114079 (ENG) |
| LXM32i Library Function Blocks Library Guide | 0198441114011 (ENG) |
| ILX Library Function blocks Software Manual | 0198441113886 (ENG) |
| Harmony XB5R ZBRN1/ ZBRN2 User Manual | EIO0000001177 (ENG) |
| Motion Control Library Guide | EIO00002221 (ENG) |

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

## Product Related Information

⚠ WARNING

LOSS OF CONTROL

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

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

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

## Terminology Derived from Standards

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

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

Among others, these standards include:

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

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

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

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

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

# Chapter 1
# Introduction

## Device Modules

### What are Device Modules

Device Modules are application code templates that provide a quick and efficient way to integrate field devices such as variable speed drives (VSD) or servo drives in the SoMachine project. The Device Modules are implemented on function templates, a mechanism within SoMachine to recall predefined application program contents.

Each Device Module contains the required SoMachine application content to control the field device, to monitor its status, and to handle errors that are detected. It includes a separate global variable list to access the different device functionalities available to the entire SoMachine automation project.

Device Modules are available for many field devices, either connected to the control system via fieldbus or hardwired. In addition, functional components that are directly associated to field devices are also contained within the Device Modules.

### How to Work with Device Modules

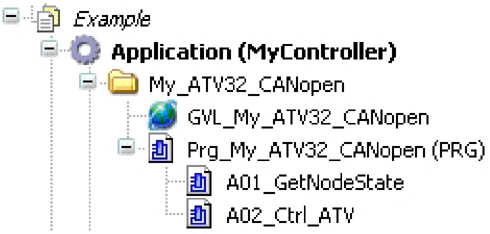
Device Modules are represented in SoMachine as code segments within a function template library.

All Device Modules described in this document are available within the SoMachine programming environment under **Tools → Template Repository → TVDA Device Module Library**. For more information, refer to the **Managing Function Templates** *(see SoMachine, Programming Guide)* online help.

Device Modules can become integrated into the application by drag-and-drop in one step. To achieve this, open the **Software Catalog** in the SoMachine Logic Builder. From the tab **Macros** in the **Software Catalog**, you can drag the desired Device Modules from the **TVDA Device Module Library** and drop it on the **Applications tree**. As a result, the **Add Function From Template** dialog box opens where you can perform the respective settings for adding the Device Modules.

**NOTE:** The name of the inserted objects corresponds with the name which has been assigned to the Device Module during the dialog **Add Function From Template** *(see SoMachine, Programming Guide)*.

Each Device Module provides a set of application objects. For a clear separation within the project, all Device Module application content inserted appears grouped in the **Application tree**:



Example
Application (MyController)
My_ATV32_CANopen
GVL_My_ATV32_CANopen
Prg_My_ATV32_CANopen (PRG)
A01_GetNodeState
A02_Ctrl_ATV

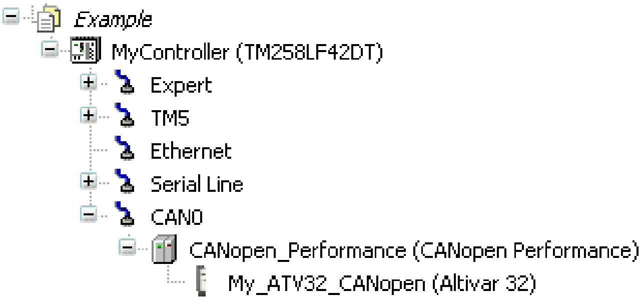
The Program (PRG) as part of a Device Module is added automatically to the task configuration of the project.

The variables (interface) of the Device Modules are declared in a global variable list (GVL). They are accessible in the project as described in the following example.

To access to the variable which indicates the communication state of the Altivar32 with the assigned instance name `My_ATV32`, you can write the following code:

```
IF NOT(GVL_My_ATV32_CANopen.xComOk)THEN
(*your program code*);
END_IF
```

Typically Device Modules include a field device. These devices are added under the associated fieldbus manager. This assumes that the respective fieldbus manager has been configured in the project before you can instantiate a Device Module. For example, the Device Module ATV32_CANopen requires a configured CANopen manager.



Example
MyController (TM258LF42DT)
Expert
TM5
Ethernet
Serial Line
CAN0
CANopen_Performance (CANopen Performance)
My_ATV32_CANopen (Altivar 32)

# Chapter 2
## Device Modules Descriptions

### What Is in This Chapter?

This chapter contains the following sections:

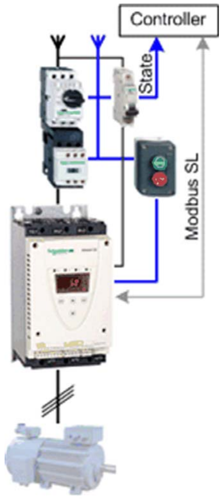
# Section 2.1
## ATS22_ModbusSL Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## ATS22_ModbusSL Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control an Altistart 22 soft start - soft stop unit via Modbus SL through a SoMachine controller.

The Device Module ATS22_ModbusSL is represented by a function template and consists of a global variable list (GVL), and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control a soft start - soft stop with the Altistart 22.

After instantiation, a variable `wModbusToken` is added to a global variable list with the name GVL. In the program, when the `wModbusToken` variable is equal to zero, the communication can start. When the communication starts, the used slave address is written to the variable. When the communication is finished, the value 0 is written to the variable. Use this variable to organize other Modbus SL communication function blocks in your application.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device in auto mode
- control the device in manual mode
- control the device in local mode

# Required Libraries

## Required Libraries Used in the ATS22_ModbusSL Device Module

The following function blocks are used in the program organization units (POU) of the Device Module. The corresponding libraries are added to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| `Mot2D1S` | TeSys Library | SE_TESYS | Schneider Electric |
| `READ_VAR` | PLCCommunication | SEN | Schneider Electric |
| `WRITE_VAR` | | | |

## Global Variable List - GVL_<name device module>

### Global Variables Provided by the ATS22_ModbusSL Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xSelAutMode | BOOL | Selects auto mode for the FB. |
| xSelManMode | BOOL | Selects manual mode for the FB. |
| xCmdLocFwd | BOOL | Local start (latch mode) of the motor in a forward direction during manual mode. |
| xCmdLocStop | BOOL | Local stop of the motor during manual mode. |
| xCmdManFwd | BOOL | Starts (latch mode) the motor during manual mode. |
| xCmdManStop | BOOL | Stops the motor during manual mode. |
| xCmdAutFwd | BOOL | Starts (jog mode) the motor during auto mode. |
| xCmdErrRst | BOOL | Resets the FB in case of an alarm state. |
| xExtLock | BOOL | External signal to lock the FB (for example state of the emergency stop). |
| xExtErr | BOOL | External signal to set the FB into error detected state (reset required). |
| xStatAutMode | BOOL | FB is selected for auto mode. |
| xStatManMode | BOOL | FB is selected for manual mode. |
| xStatErr | BOOL | FB is in error state, reset required. |
| xDriveCmdFrwhStop | BOOL | Activates the freewheel deceleration within the control word xDriveCmdRun. |
| xDriveCmdEn2ndPara | BOOL | Enables the second set of parameters. |
| xDriveCmdLocMode | BOOL | Forces local commands directly on the drive. Not related to the local commands used in the program. |
| xDriveRst | BOOL | Resets the drive in case of an error state. |
| xMcbRdy | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the respective auxiliary contact of the MCB is connected. |
| xDriveRun | BOOL | Signal associated with the status word which indicates the motor is energized through the Altistart 22. |
| xDriveRdy | BOOL | Signal associated with the status word which indicates the Altistart 22 is operational. |
| xDriveTrip | BOOL | Signal associated with the status word which indicates whether an error has been detected on the Altistart 22. |
| xDriveWarn | BOOL | Signal associated with the status word which indicates whether an advisory condition exists on the Altistart 22. |

| Variable | Data Type | Description |
|---|---|---|
| `xDriveLocMode` | BOOL | Signal associated with the status word which indicates whether the local command is forced on the Altistart 22. |
| `xDriveRamping` | BOOL | Signal associated with the status word which indicates whether the motor is accelerating or decelerating. |
| `byDriveMotCurr` | BYTE | Signal associated with the status word which indicates the present motor current in percent of the rated motor current.<br>Range: 0...200% |
| `xAlertLock` | BOOL | FB is blocked by `i_xLock`. |
| `xAlarmFbckTmout` | BOOL | FB is in alarm state due to undetected feedback signal during the monitoring time. |
| `xAlarmExt` | BOOL | FB is in alarm state due to detected `i_xErr`. |
| `xComOk` | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| `byComErrRead` | BYTE | Indicates the error ID in case of a detected communication error during read request. |
| `dwOperErrRead` | DWORD | Indicates the error ID in case of a detected operation error during read request. |
| `byComErrWrite` | BYTE | Indicates the error ID in case of a detected communication error during write request. |
| `dwOperErrWrite` | DWORD | Indicates the error ID in case of a detected operation error during write request. |
| `xComInit` | BOOL | Indicates that the communication has been successfully intialized. |

## Program - Prg_<name device module>

### Program Contained in the ATS22_ModbusSL Device Module

The program is divided into 3 actions and is created in programming language CFC (Continuous Function Chart). All actions will be called on each program execution.

### Action - A01_ReadParameter

By the program code in this action, the status word of the device is read via Modbus SL using the FB `READ_VAR`.

The Modbus communication is managed internally with the variable `GVL.wModbusToken`. The token coordinates the execution of all Modbus function blocks configured in the project. The token is being reserved by the active Modbus function block until the Modbus operation has been completed. If the read process of the status word is possible, the variable `GVL.wModbusToken` will be reserved (the slave address is written on it), until the request is completed.

The value of the status word is assigned to the corresponding variables which have been declared in the associated GVL *(see page 24)*.

### Action - A02_AltistartControl

By the program code in this action, the motor control FB is called.

Implemented features are:
- Mapping of the manual commands to the control word.
- FB instance (`MOT1D1S`) call with assigned parameters.
- Extracting of the status word (detailed alarm and alert information) to boolean variables.

### Action - A03_WriteParameter

By the program code in this action the control word of the device is written via Modbus SL using the FB `WRITE_VAR`.

The Modbus communication is managed internally with the variable `GVL.wModbusToken`. The token controls execution of all Modbus function blocks configured in the project. The token is being reserved by the active Modbus function block until the Modbus operation has been completed. If the write process of the control word is possible, the `GVL.wModbusToken` will be reserved (the slave address is written on it) until the request is completed.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires that a Modbus manager be added to the serial interface of your controller.

Using **Add Function from Template** *(see SoMachine, Programming Guide)* for this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variable selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdy` | BOOL | – | Signal associated with the motor circuit breaker contact indicating that the device is under power. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.c_bywAddr` | BYTE | 1 | Modbus slave address of the device. |
| `Prg_<module name>.c_byChanNb` | BYTE | 1 | Communication port of the controller. |
| `Prg_<modul name>.c_xEnFbckCtrl` | BOOL | TRUE | Enables the monitoring of the feedback signals of the motor run state. |
| `Prg_<modul name>.c_iDlyTimeFbckCtrl` | INT | 2 | Delay time in seconds to determine that the feedback signal is inoperable and to activate an alarm. |

# Section 2.2
## ATV•••_EtherNet/IP Device Modules

### Overview

This section provides a generic description for the following Device Modules:

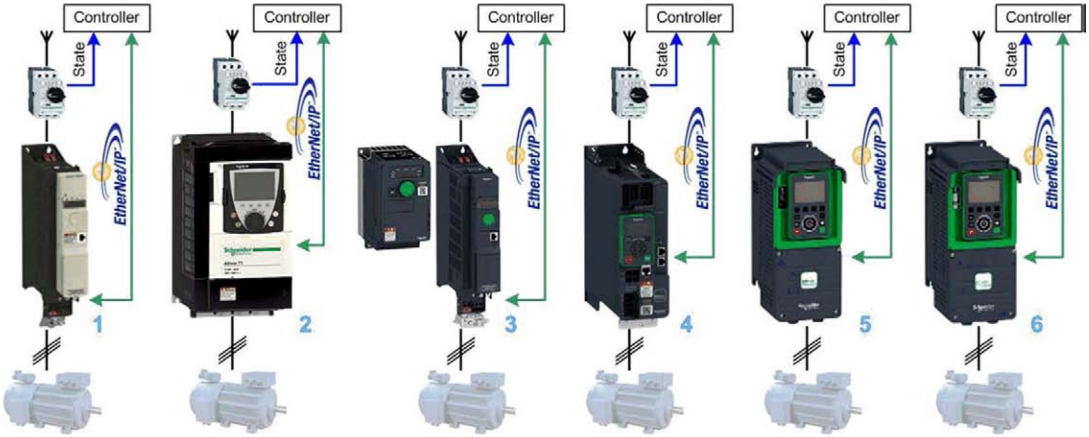
- ATV32_EtherNetIP
- ATV71_EtherNetIP
- ATV320_EtherNetIP
- ATV340_EtherNetIP
- ATV6xx_EtherNetIP
- ATV9xx_EtherNetIP

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



1 Altivar 32
2 Altivar 71
3 Altivar 320 - Compact and Book format
4 Altivar 340
5 Altivar 6••
6 Altivar 9••

## Device Module Description

Each Device Module covered by this description provides the application objects and the device which are required to monitor and control the associated Altivar type via EtherNet/IP with a Schneider Electric SoMachine controller. Each device (Altivar) requires the **Industrial Ethernet manager** under the Ethernet interface of the controller within the **Devices tree** of the Logic Builder configuration.

# Required Libraries

## Required Libraries Used in a Device Module

A Device Module implements objects from one or more libraries. The objects and the associated libraries are listed in the following tables.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power | GMC Independent PLCopen MC | GIPLC | Schneider Electric |
| MC_Reset | | | |
| MC_Stop | | | |
| MC_Jog | | | |
| MC_MoveVelocity | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadStatus | | | |
| MC_ReadAxisError | | | |
| SetDriveRamp_ATV | GMC Independent Altivar | GIATV | |
| SetFrequencyRange_ATV | | | |
| StoreParameters_ATV | | | |
| EIPGetHealthBit | EtherNetIP Scanner | EIPSC | |
| EIPStartConnection | | | |
| EIPStopConnection | | | |
| FB_RemoteAdapter | EtherNetIP Remote Adapter | EIPRA | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| OperationErrorCodes | EtherNetIP Scanner | EIPSC | Schneider Electric |
| CIPOperationErrorCodes | | | |
| CommunicationErrorCodes | | | |
| eStatus | EtherNetIP Remote Adapter | EIPRA | |
| eAdapterErrorInfo | | | |

**NOTE:** The library EtherNetIP Scanner is not supported by the motion controller LMC078.

## Functional Description

### Device - <name device module>

Each Device Module implements the device for the associated Altivar type for EtherNet/IP. The device is added under the **Industrial Ethernet manager** in your configuration with the instance name assigned within the **Add Function From Template** dialog box.

The device is preconfigured. The configuration includes the connection for the native drive control profile with the assemblies 100 (output) and 101 (input).

The Requested Packet Interval (RPI) is configured to:

- 10 ms for the Altivar 32, 320, 340, 6••, and 9••
- 30 ms for the Altivar 71

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node in the folder with the name assigned within the **Add Function From Template** dialog box. The GVL has the same name as the device instance with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name assigned within the **Add Function From Template** dialog box. The program has the same name as the device instance with the prefix `Prg_`. A program-call is added automatically to the associated task.

For basic control functions, the program code does not need to be modified, all required signals and control parameter are linked to the associated variables in the GVL.

The program is divided into several actions. These are described in the following table.

**NOTE:** The program logic of the action `A01_ComCtrl` is not supported in an application of a motion controller LMC078.

| Name of the action | Description |
|---|---|
| A01_ComCtrl | Processes the functions to monitoring and control of the EtherNet/IP communication with the device. |
| A02_Ctrl_ATV | Contains a selection of function block calls to control and monitor the Altivar. Each function block is called in each program cycle. |
| A03_Config_ATV | Contains a selection of function block calls to write a set of parameters to the Altivar. |
| A04_FbErrorDetection | Contains the logic for the evaluation of the error messages which are provided by the motion control (MC_) function blocks. |

Further information about the control logic is available inside the program in terms of comments.

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module requires the **Industrial Ethernet manager** under the Ethernet interface of your controller within the **Devices tree** of the Logic Builder configuration.

Using **Add Function From Template** you can:

- Select the fieldbus master which manages the device
- Assign the IP address for the device
- Map variables to physical inputs and outputs of your configuration
- Adjust initial values for selected variables which are part of the template

Variable selected for I/O mapping (input):

| Variable | Data type | Default value | Description |
|---|---|---|---|
| `GVL_<name device module>.xMcbRdy` | BOOL | – | Indicates the state of motor circuit breaker. |

# Section 2.3
## ATV•••_ModbusTCP Device Modules

### Overview

This section provides a generic description for the following Device Modules:

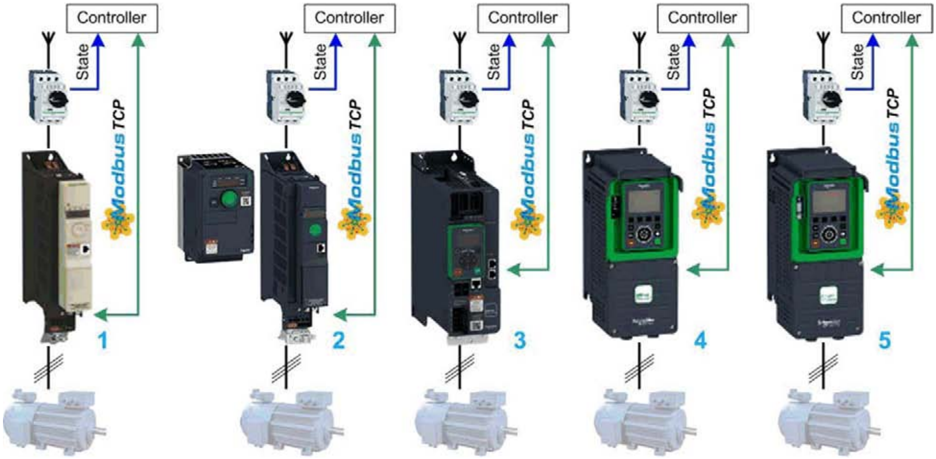
- ATV32_ModbusTCP
- ATV320_ModbusTCP
- ATV340_ModbusTCP
- ATV6xx_ModbusTCP
- ATV9xx_ModbusTCP

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



1 Altivar 32
2 Altivar 320 - Compact and Book format
3 Altivar 340
4 Altivar 6••
5 Altivar 9••

## Device Module Description

Each Device Module covered by this description provides the application objects and the device which are required to monitor and control the associated Altivar type via Modbus TCP with a Schneider Electric SoMachine controller. Each device (Altivar) requires the **Industrial Ethernet manager** under the Ethernet interface of the controller within the **Devices tree** of the Logic Builder configuration.

# Required Libraries

## Required Libraries Used in a Device Module

A Device Module implements objects from one or more libraries. The objects and the associated libraries are listed in the following table.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power | GMC Independent PLCopen MC | GIPLC | Schneider Electric |
| MC_Reset | | | |
| MC_Stop | | | |
| MC_Jog | | | |
| MC_MoveVelocity | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadStatus | | | |
| MC_ReadAxisError | | | |
| SetDriveRamp_ATV | GMC Independent Altivar | GIATV | |
| SetFrequencyRange_ATV | | | |
| StoreParameters_ATV | | | |
| IOS_GetHealth | ModbusTCPIOScanner | SE_IOS | |

## Functional Description

### Device - <name device module>

Each Device Module implements the device for the associated Altivar type for Modbus TCP. The device is added under the **Industrial Ethernet manager** in your configuration with the instance name assigned within the **Add Function From Template** *(see SoMachine, Programming Guide)* dialog box.

The device is preconfigured. The configuration includes the **ModbusTCP channel** for the cyclic data exchange with the device. The repetition rate for the channel is selected with 10 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node in the folder with the name assigned within the **Add Function From Template** dialog box. The GVL has the same name as the device instance with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name assigned within the **Add Function From Template** dialog box. The program has the same name as the device instance with the prefix `Prg_`. A program-call is added automatically to the associated task.

For basic control functions, the program code does not need to be modified, all required signals and control parameter are linked to the associated variables in the GVL.

The program is divided into several actions. These are described in the following table.

| Name of the action | Description |
|---|---|
| A01_ComStat | Processes the functions to monitoring and control of the Modbus TCP communication with the device. |
| A02_Ctrl_ATV | Contains a selection of function blocks calls to control and monitor the Altivar. Each function block is called in each program cycle. |
| A03_Config_ATV | Contains a selection of function blocks calls to write a set of parameters to the Altivar. |
| A04_FbErrorDetection | Contains the logic for the evaluation of the error messages which are provided by the motion control (MC_) function blocks. |

**NOTE:** For monitoring the communication state of the device, the channel ID of the configured Modbus TCP channel must be set as value for the variable `GVL_<name device module>.c_uiChannelId`. The channel ID is automatically generated when the device is added to the project and can be obtained through the **Device Editor** in the tab **Modbus TCP Slave configuration**.
Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires the **Industrial Ethernet manager** under the Ethernet interface of your controller within the **Devices tree** of the Logic Builder configuration.

Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can:
- Select the fieldbus master which manages the device
- Assign the IP address for the device
- Map variables to physical inputs and outputs of your configuration
- Adjust initial values for selected variables which are part of the template

Variable selected for I/O mapping (input):

| Variable | Data type | Default value | Description |
|---|---|---|---|
| `GVL_<name device module>.xMcbRdy` | BOOL | – | Indicates the state of motor circuit breaker. |

# Section 2.4
## ATV•••_CANopen Device Modules

### Overview

This section provides a generic description for the following Device Modules:

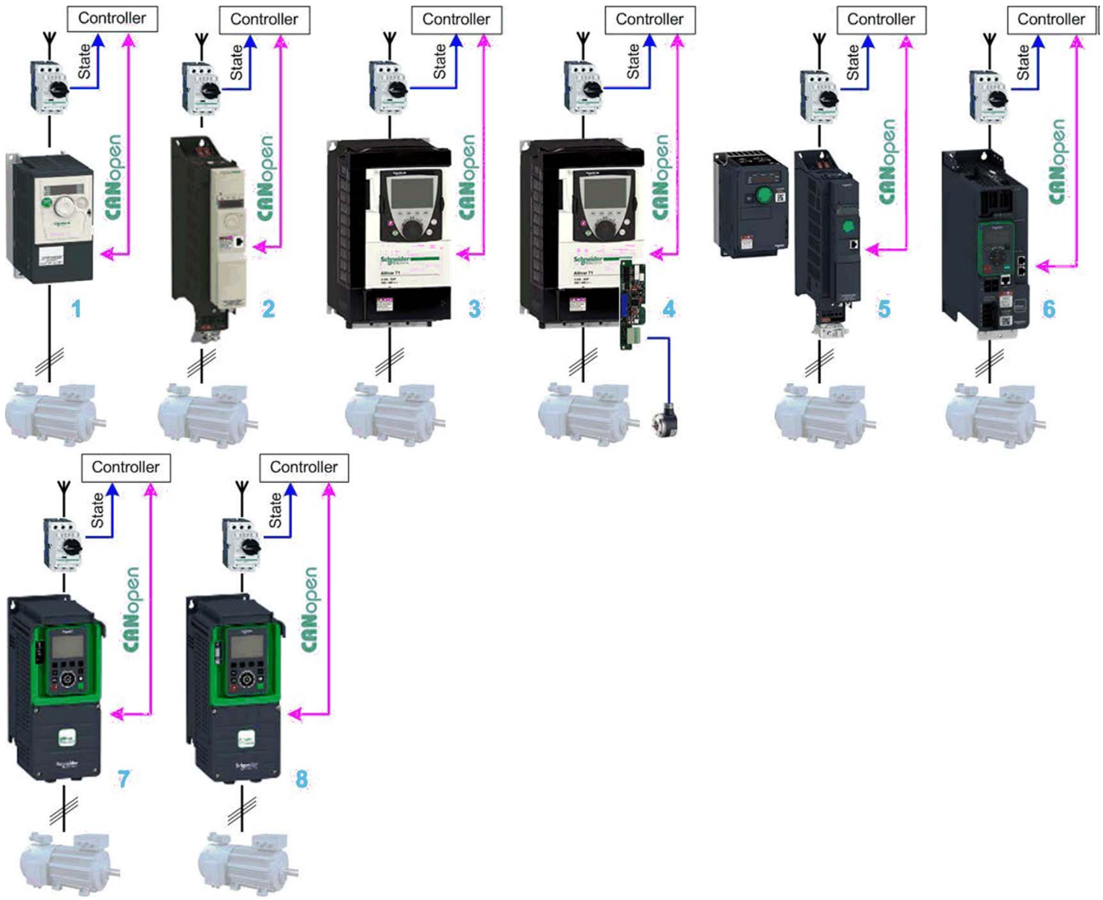
- ATV312_CANopen
- ATV32_CANopen
- ATV71_CANopen
- ATV71_CANopen_Enc
- ATV320_CANopen
- ATV340_CANopen
- ATV6xx_CANopen
- ATV9xx_CANopen

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



1 ATV312_CANopen
2 ATV32_CANopen
3 ATV71_CANopen
4 ATV71_CANopen_Enc
5 ATV320_CANopen
6 ATV340_CANopen
7 Altivar 6••
8 Altivar 9••

## Device Module Description

Each Device Module covered by this description provides the application objects and the device which are required to monitor and control the associated Altivar type via CANopen with a Schneider Electric SoMachine controller. Each device (Altivar) requires the CANopen manager under the CAN interface of the controller within the **Devices tree** of the Logic Builder configuration.

# Required Libraries

## Required Libraries Used in a Device Module

A Device Module implements objects from one or more libraries.

With the SoMachine V4.1 SP2, a new library concept was introduced for devices that comply with the PLCopen standard. Therefore, the motion control (MC_) function blocks for the Altivar 320, 340, 6••, and 9•• devices are provided in a new, fieldbus-independent library.

The objects and the associated libraries are listed in the following tables.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| Used only by the Device Modules ATV320_CANopen, ATV340_CANopen, ATV6xx_CANopen, and ATV9xx_CANopen. | | | |
| MC_Power | GMC Independent PLCopen MC | GIPLC | Schneider Electric |
| MC_Reset | | | |
| MC_Stop | | | |
| MC_Jog | | | |
| MC_MoveVelocity | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadStatus | | | |
| MC_ReadAxisError | | | |
| SetDriveRamp_ATV | GMC Independent Altivar | GIATV | |
| SetFrequencyRange_ATV | | | |
| StoreParameters_ATV | | | |
| Used only by the Device Modules ATV312_CANopen, ATV32_CANopen,ATV71_CANopen, and ATV71_CANopen_Enc. | | | |
| MC_Power_ATV | Altivar Library | SE_ATV | Schneider Electric |
| MC_Reset_ATV | | | |
| MC_Stop_ATV | | | |
| MC_Jog_ATV | | | |
| MC_MoveVelocity_ATV | | | |
| MC_ReadAxisError_ATV | | | |
| SetDriveRamp_ATV | | | |
| SetFrequencyRange_ATV | | | |
| StoreParameters_ATV | | | |
| Used by the Device Modules supported in this description. | | | |
| GET_STATE | CAA CiA405 | CIA405 | CAA Technical Workgroup |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| Used by the Device Modules supported in this description. | | | |
| `DEVICE_STATE` | CAA CiA405 | CIA405 | CAA Technical Workgroup |

## Functional Description

### Device - <name device module>

Each Device Module implements the device for the associated Altivar type for CANopen. The device is added under the CANopen manager in your configuration with the instance name assigned within the **Add Function From Template** dialog box.

The device is preconfigured and corresponds to the default configuration of the associated device except for the device Altivar 71 as part of the Device Module ATV71_CANopen_Enc.

The Device Module ATV71_CANopen_Enc supports an application where an encoder is linked to the drive Altivar 71 and the encoder value is read via CANopen from the controller. Therefore, the object **PUC** (pulse counter) is added to the first transmit PDO (Process Data Object) in the device editor of the Altivar 71. The resulting additional entry in the input mapping table represents the encoder value and is assigned to a variable in the associated GVL for monitoring purposes only.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node in the folder with the name assigned within the **Add Function From Template** dialog box. The GVL has the same name as the device instance with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a POU of type **Program**. This program is added under the **Application** node within a folder with the name assigned within the **Add Function From Template** dialog box. The program has the same name as the device instance with the prefix `Prg_`. A program-call is added automatically to the associated task.

For the main control functions, the program code does not need to be modified, the required signals and control parameters are linked to the associated variables in the GVL.

The program is divided into several actions. They are described in the following table.

| Name of the action | Description |
|---|---|
| A01_GetNodeState | Contains the call of the GET_STATE function block to obtain the communication state for the CANopen device. |
| A02_Ctrl_ATV | Contains a selection of function block calls to control and monitor the Altivar. Each function block is called in each program cycle. |
| A03_Config_ATV | Contains a selection of function block calls to write a set of parameters to the Altivar. |
| A04_FbErrorDetection | Contains the logic for the evaluation of the error messages which are provided by the motion control (MC_) function blocks. |

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires the CANopen manager under the CAN interface of the controller within the **Devices tree** of the Logic Builder configuration.

Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can:

- Select the fieldbus master which manages the device.
- Assign the node ID to the CANopen remote device.
- Map variables to physical inputs and outputs of your configuration.
- Adjust initial values for selected variables which are part of the template.

Variable selected for I/O mapping (input):

| Variable | Data type | Default value | Description |
|---|---|---|---|
| `GVL_<name device module>.xMcbRdy` | BOOL | – | Indicates the state of motor circuit breaker. |

Variable selected for parameterization (constant):

| Variable | Data type | Default value | Description |
|---|---|---|---|
| `Prg_<name device module>.c_udiTmotGetStat` | UDINT | 1000 | Parameter for timeout monitoring on FB instance `GET_STATE` in ms. |

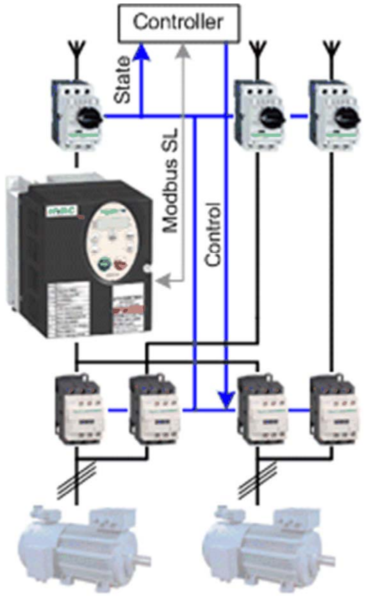
# Section 2.5
## ATV212_ModbusSL_2Motors_Bypass Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## ATV212_ModbusSL_2Motors_Bypass Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control two motors. Each motor can be controlled either via an Altivar 212 variable speed drive, or with a direct online motor starter to bypass the drive. Only one motor can be controlled via the drive at the same time. The Altivar 212 is controlled and monitored via Modbus SL and the direct online motor starters are controlled and monitored via hardwired signals through a SoMachine controller.

The Device Module ATV212_ModbusSL_2Motors_Bypass is represented by a function template and consists of a global variable list (GVL), and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control an ATV212, the switching between the motors, and the bypass control of the motors.

After instantiation, a variable `wModbusToken` is added a global variable list with the name GVL. In the program, when the `wModbusToken` variable is equal to zero, the communication can start. When the communication starts, the used slave address is written to the variable. When the communication is finished, the value 0 is written to the variable. Use this variable to organize other Modbus SL communication function blocks in your application.

The program provides the following features:

- monitor the communication state of the drive
- monitor the state of the drive and the direct online motor starters
- control both motors in auto mode
- control both motors in manual mode
- control both motors in local mode
- control both motors in bypass mode
- control one of the motors via the drive
- reset the drive in case an error state is detected

# Required Libraries

## Required Libraries Used in the ATV212_ModbusSL_2Motors_Bypass Device Module

The following function blocks are used in the program organization units (POU) of the Device Module. The corresponding libraries are added to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| Mot2D1S | TeSys library | SE_TESYS | Schneider Electric |
| READ_VAR | PLCCommunication | SEN | Schneider Electric |
| WRITE_VAR | | | |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the ATV212_ModbusSL_2Motors_Bypass Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xMcbRdyDrive | BOOL | Indicates the state of the Motor Circuit Breaker (MCB). Mapped to the physical input where the respective auxiliary contact of the MCB is connected. |
| xMcbRdyBypass1 | BOOL | Indicates the state of the MCB of the direct online motor starter of motor 1. Mapped to the physical input where the respective auxiliary contact of the MCB is connected. |
| xMcbRdyBypass2 | BOOL | Indicates the state of the MCB of the direct online motor starter of motor 2. Mapped to the physical input where the respective auxiliary contact of the MCB is connected. |
| xContactorActvMot1 | BOOL | Indicates one of the contactors of motor 1 is energized. |
| xContactorActvMot2 | BOOL | Indicates one of the contactors of motor 2 is energized. |
| xSelBypassModeMot1 | BOOL | Pre-selection for control of motor 1 via the direct online motor starter. |
| xSelBypassModeMot2 | BOOL | Pre-selection for control of motor 2 via the direct online motor starter. |
| xSelDriveCtrlModeMot1 | BOOL | Pre-selection for control of motor 1 via the drive. |
| xSelDriveCtrlModeMot2 | BOOL | Pre-selection for control of motor 2 via the drive. |
| xSelAutModeMot1 | BOOL | Selects auto mode for the FB of motor 1. |
| xSelAutModeMot2 | BOOL | Selects auto mode for the FB of motor 2. |
| xSelManModeMot1 | BOOL | Selects manual mode for the FB of motor 1. |
| xSelManModeMot2 | BOOL | Selects manual mode for the FB of motor 2. |
| xCmdManFwdMot1 | BOOL | Starts (latch mode) the motor 1 in a forward direction during manual mode. |
| xCmdManRevMot1 | BOOL | Starts (latch mode) the motor 1 in a reverse direction during manual mode. |
| xCmdManStopMot1 | BOOL | Stops the motor 1 during manual mode. |
| xCmdManFwdMot2 | BOOL | Starts (latch mode) the motor 2 in a forward direction during manual mode. |
| xCmdManRevMot2 | BOOL | Starts (latch mode) the motor 2 in a reverse direction during manual mode. |
| xCmdManStopMot2 | BOOL | Stops the motor 2 during manual mode. |

| Variable | Data Type | Description |
|---|---|---|
| xCmdLocFwdMot1 | BOOL | Local start (latch mode) of the motor 1 in a forward direction during manual mode. |
| xCmdLocRevMot1 | BOOL | Local start (latch mode) of the motor 1 in a reverse direction during manual mode. |
| xCmdLocStopMot1 | BOOL | Local stop of the motor 1 during manual mode. |
| xCmdLocFwdMot2 | BOOL | Local start (latch mode) of the motor 2 in a forward direction during manual mode. |
| xCmdLocRevMot2 | BOOL | Local start (latch mode) of the motor 2 in a reverse direction during manual mode. |
| xCmdLocSTopMot2 | BOOL | Local stop of the motor 2 during manual mode. |
| xCmdAutFwdMot1 | BOOL | Starts (jog mode) the motor 1 in a forward direction during auto mode. |
| xCmdAutRevMot1 | BOOL | Starts (jog mode) the motor 1 in a reverse direction during auto mode. |
| xCmdAutFwdMot2 | BOOL | Starts (jog mode) the motor 2 in a forward direction during auto mode |
| xCmdAutRevMot2 | BOOL | Starts (jog mode) the motor 2 in a reverse direction during auto mode. |
| xCmdErrRstMot1 | BOOL | Resets the FB controlling motor 1 in case of an alarm state. |
|  | BOOL | Resets the FB controlling of motor 2 in case of an alarm state. |
|  | BOOL | External signal to set the FB controlling motor 1 (for example state of the emergency stop). |
|  | BOOL | External signal to set the FB controlling motor 2 (for example state of the emergency stop). |
|  | BOOL | External signal to set the FB controlling motor 1, into error detected state (reset required). |
|  | BOOL | External signal to set the FB controlling motor 2, into error detected state (reset required). |
|  | BOOL | Activates the contactor of the direct online motor starter of motor 1. |
|  | BOOL | Activates the contactor of the direct online motor starter of motor 2. |
| xCmdErrRstMot2 | BOOL | Activates the contactor which links the motor 1 to the drive. |
| xExtLockMot1 | BOOL | Activates the contactor which links the motor 2 to the drive. |
| xExtLockMot2 | WORD | Speed reference for manual mode associated to the FB controlling motor 1 in units of 0.01 Hz. |

| Variable | Data Type | Description |
|---|---|---|
| xExtErrMot1 | WORD | Speed reference for automatic mode associated to the FB controlling motor 1 in units of 0.01 Hz. |
| xExtErrMot2 | WORD | Speed reference for manual mode associated to the FB controlling motor 2 in units of 0.01 Hz. |
| xCmdActvBypassContactMot1 | WORD | Speed reference for automatic mode associated to the FB controlling motor 2 in units of 0.01 Hz. |
| xCmdActvBypassContactMot2 | BOOL | Indicates motor 1 is selected for Bypass mode |
| xStatBypassModeMot2 | BOOL | Indicates motor 2 is selected for Bypass mode. |
| xStatDriveCtrlModeMot1 | BOOL | Indicates motor 1 is selected for ATV control mode. |
| xStatDriveCtrlModeMot2 | BOOL | Indicates motor 2 is selected for ATV control mode. |
| xMot1RunBypassMode | BOOL | Indicates whether motor 1 is running in Bypass mode. |
| xMot1RunDriveMode | BOOL | Indicates whether motor 1 is running in ATV control mode. |
| xMot2RunBypassMode | BOOL | Indicates whether motor 2 is running in Bypass mode. |
| xMot2RunDriveMode | BOOL | Indicates whether motor 2 is running in ATV control mode. |
| xStatAutModeMot1 | BOOL | FB controlling motor 1 is selected for auto mode. |
| xStatManModeMot1 | BOOL | FB controlling motor 1 is selected for manual mode. |
| xStatLocModeMot1 | BOOL | FB controlling motor 1 is selected for local mode. |
| xStatAutModeMot2 | BOOL | FB controlling motor 2 is selected for auto mode. |
| xStatManModeMot2 | BOOL | FB controlling motor 2 is selected for manual mode. |
| xStatLocModeMot2 | BOOL | FB controlling motor 2 is selected for local mode. |
| xStatErrMot1 | BOOL | FB controlling motor 1 is in error detected state, reset required. |
| xStatErrMot2 | BOOL | FB controlling motor 2 is in error detected state, reset required. |
| xAlertLockMot1 | BOOL | FB controlling motor 1 is blocked by i_xLock. |
| xAlertLockMot2 | BOOL | FB controlling motor 2 is blocked by i_xLock. |
| xAlarmOpModeMot1 | BOOL | Invalid operation mode selection received for motor 1. |
| xAlarmOpModeMot2 | BOOL | Invalid operation mode selection received for motor 2. |
| xAlarmExtMot1 | BOOL | FB controlling motor 1 is in alarm state due to detected i_xErr. |
| xAlarmExtMot2 | BOOL | FB controlling motor 2 is in alarm state due to detected i_xErr. |
| xAlarmFbckTmoutMot1 | BOOL | FB controlling motor 1 is in alarm state due to undetected feedback signal (i_xFwdFbck or i_xRevFbck) during the monitoring time. |

| Variable | Data Type | Description |
|---|---|---|
| `xAlarmFbckTmoutMot2` | BOOL | FB controlling motor 2 is in alarm state due to undetected feedback signal (`i_xFwdFbck` or `i_xRevFbck`) during the monitoring time. |
| `xDriveForceDcBrake` | BOOL | Signal associated with the command word to force the DC braking. |
| `xDriveCmdEStop` | BOOL | Signal associated with the command word to initiate an emergency stop on the drive. |
| `xDriveRst` | BOOL | Signal associated with the command word to reset the drive 212 in case of alarm state. |
| `xDriveTrip` | BOOL | Signal associated with the status word indicating whether an error has been detected on the drive. |
| `xDriveAlarm` | BOOL | Signal associated with the status word which indicates whether an alarm condition exists on the drive. |
| `xDriveMcUVltgAlarm` | BOOL | Signal associated with the status word which indicates whether an under-voltage condition exists on the drive. |
| `xDriveDcBrakeForced` | BOOL | Signal associated with the status word which indicates the DC braking is forced. |
| `xDriveRunDir` | BOOL | Signal associated with the status word which indicates the direction when the drive is running. (FALSE = forward, TRUE = reverse) |
| `xDriveEStopActv` | BOOL | Signal associated with the status word which indicates an emergency stop state for the drive. |
| `xPtcMot1Ok` | BOOL | Indicates a detected overheating of motor 1. Mapped to the physical input where the respective signaling contact is connected. |
| `xPtcMot2Ok` | BOOL | Indicates a detected overheating of motor 2. Mapped to the physical input where the respective signaling contact is connected. |
| `xComOk` | BOOL | Indicates the communication state of the drive.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| `byComErrRead` | BYTE | Indicates the error ID in case of a detected communication error during read request. |
| `dwOperErrRead` | DWORD | Indicates the error ID in case of a detected operation error during read request. |
| `byComErrWrite` | BYTE | Indicates the error ID in case of a detected communication error during write request. |
| `dwOperErrWrite` | DWORD | Indicates the error ID in case of a detected operation error during write request. |
| `xComInit` | BOOL | Indicates the communication state for the drive. |

## Program - Prg_<name device module>

### Program Contained in the ATV212_ModbusSL_2Motors_Bypass Device Module

The program is divided into 6 actions and is created in programming language CFC. (continuous function chart). All actions will be called on each program execution.

### Action - A01_ReadParameter

By the program code in this action the status word of the device is read via Modbus SL using the FB `READ_VAR`.

The Modbus communication is managed internally with the variable `GVL.wModbusToken`. The token is used to control the execution of all Modbus function blocks configured in the project. The token is being reserved by the active Modbus function block until the Modbus operation has been completed. If the read process of the status word is possible, the variable `GVL.wModbusToken` will be reserved (the slave address is written on it), until the request is completed.

The value of the status word is assigned to corresponding variables which have been declared in the GVL *(see page 52)*.

### Action - A02_ContactorControl

The program code in this action controls the contactors for selection if a motor shall be controlled either by the drive or by the direct online motor starter. The logic in this Device Module allows only one motor controlled by the drive at the same time. The commands for selecting the different control modes for both motors are assigned to corresponding variables which have been declared in the `GVL_<module name>.`

### Action - A03_Motor1Control

By the program code in this action the motor control FB for motor 1 is called.

Implemented features are:

- Mapping the manual commands into the control word.
- FB instance (`MOT2D1S`) call with assigned parameters.
- Extracting of the status word (detailed alarm and alert information) to boolean variables.

### Action - A04_Motor2Control

By the program code in this action the motor control FB for motor 2 is called.

Implemented features are:

- Mapping the manual commands into the control word.
- FB instance (`MOT2D1S`) call with assigned parameters.
- Extracting of the status word (detailed alarm and alert information) to boolean variables.

## Action - A05_WriteParameter

By the program code in this action the control word of the device is written via Modbus SL using the FB `WRITE_VAR`.

The Modbus communication is managed internally with the variable `GVL.wModbusToken`. The token is used to control the execution of all Modbus function blocks configured in the project. The token is being reserved by the active Modbus function block until the Modbus operation has been completed. If the write process of the control word is possible, the `GVL.wModbusToken` will be reserved (the slave address is written on it) until the request is completed.

## Action - A06_CheckState

By the program code in this action, the Modbus communication with the Altistart and the status of motor 1 and motor 2 is monitored.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires that a Modbus manager be added to the serial interface of your controller.

Using **Add Function from Template** *(see SoMachine, Programming Guide)* for this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdyDrive` | BOOL | – | Indicates the state of Motor Circuit Breaker (MCB). |
| `GVL_<modul name>.xMcbRdyBypass1` | BOOL | – | Indicates the state of the MCB of the direct online motor starter of motor 1. |
| `GVL_<modul name>.xMcbRdyBypass2` | BOOL | – | Indicates the state of the MCB of the direct online motor starter of motor 2. |
| `GVL_<modul name>.xContactorActvMot1` | BOOL | – | Indicates one of the contactors of motor 1 is energized. |
| `GVL_<modul name>.xContactorActvMot2` | BOOL | – | Indicates one of the contactors of motor 2 is energized. |

Variables selected for I/O mapping (output):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xCmdActvBypassContactMot1` | BOOL | – | Activates the contactor of the direct online motor starter of motor 1. |
| `GVL_<modul name>.xCmdActvBypassContactMot2` | BOOL | – | Activates the contactor of the direct online motor starter of motor 2. |
| `GVL_<modul name>.xCmdActvDriveContactMot1` | BOOL | – | Activates the contactor which links the motor 1 to the drive. |
| `GVL_<modul name>.xCmdActvDriveContactMot2` | BOOL | – | Activates the contactor which links the motor 2 to the drive. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.wAddr` | WORD | 1 | Modbus slave address. |
| `Prg_<module name>.c_byChanNb` | BYTE | 1 | Communication port of the controller. |
| `Prg_<modul name>.xEnFbckCtrlMot1` | BOOL | TRUE | Enables the monitoring of the feedback signals of the motor 1 run state. |
| `Prg_<modul name>.c_iDlyTimeFbckCtrlMot1` | INT | 2 | Delay time in seconds to determine that the feedback signal from motor 1 is inoperable and to activate an alarm. |
| `Prg_<modul name>.c_iDlyTimeRevsMot1` | INT | 2 | Delay time in seconds to invert the direction of the running motor 1. |
| `Prg_<modul name>.c_xEnFbckCtrlMot2` | BOOL | TRUE | Enables the monitoring of the feedback signals of the motor 2 run state. |
| `Prg_<modul name>.c_iDlyTimeFbckCtrlMot2` | INT | 2 | Delay time in seconds to determine that the feedback signal from motor 2 is inoperable and to activate an alarm. |
| `Prg_<modul name>.iDlyTimeRevsMot2` | INT | 2 | Delay time in seconds to invert the direction of the running motor 2. |

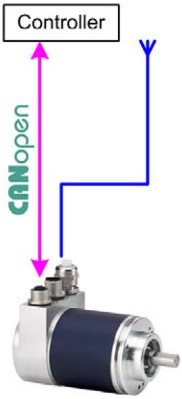
# Section 2.6
## Encoder_AbsMlt_CANopen Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Encoder_AbsMlt_CANopen Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor an absolute multiturn encoder (OsiCoder) via CANopen through a SoMachine controller.

The Device Module Encoder_AbsMlt_CANopen is represented by a function template and consists of a global variable list, a program including one action, and the device OsiCoder under the CANopen manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the OsiCoder.

The program provides the following features:
- monitor the communication state of the device

The device OsiCoder used in this Device Module differs from the standard OsiCoder device provided with the SoMachine **Device Repository**. On the used device the second transmit PDO (Process Data Object) was deactivated. For the first transmit PDO, the Event Time is set to 100 ms and the Inhibit Time is set to 10 ms. Therefore, there is a transmission between 10 ms (inhibit time) given data in the PDO has changed, and 100 ms (event time) given no data has changed.

## Required Libraries

### Required Libraries Used in the Encoder_AbsMlt_CANopen Device Module

The following function block is used in the POU of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| `Get_State` | CAA CiA 405 | CIA405 | CAA Technical Workgroup |

## Global Variable List - GVL_<name device module>

### Global Variables Provided by the Encoder_AbsMlt_CANopen Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| `udiEncVal` | UDINT | Indicates the value of the encoder. This variable is already mapped to the object **position value** in the device dialog **CANopen  I/O mapping**. |
| `xComOk` | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| `eComStat` | CIA405.DEVICE_STATE | Communication state of the device. For information on the enumeration, refer to the CIA405 Library Guide (see SoMachine Online Help under *CoDeSys Libraries/CAA Libraries/CAA_CiA405.library*). |

## Program - Prg_<name device module>

### Program Contained in the Encoder_AbsMlt_CANopen Device Module

The program is created in programming language CFC (Continuous Function Chart) and calls an action on each program execution.

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANopen fieldbus. Based on the communication parameter the communication state is assigned to the corresponding variables which have been declared in the GVL_<module name>. If the state is equal to OPERATIONAL, the variable for the state indicates TRUE and other cases are indicated by FALSE.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires that a CANopen manager be added to the CAN interface of your controller.

Using the instantiation, you can:
- select the CANopen manager which shall manage the device
- assign the CANopen node address for the device
- adjust initial values for selected variables which are part of the template

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.c_udiTmotGetStat` | UDINT | 1000 | Parameter for timeout monitoring on FB instance `GET_STATE`. |
| `Prg_<modul name>.c_uiNodeId` | UINT | 1 | Node address of the CANopen device. |
| `Prg_<modul name>.c_uiNetworkNo` | UINT | 1 | Network number on which the device is linked. |

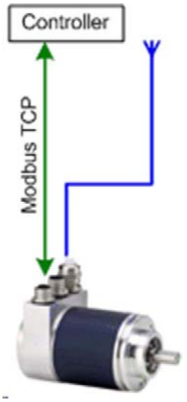
# Section 2.7
## Encoder_AbsMlt_ModbusTCP Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Encoder_AbsMlt_ModbusTCP Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor an absolute multiturn encoder via Modbus TCP through a SoMachine controller.

The Device Module Encoder_AbsMlt_ModbusTCP is represented by a function template and consists of a global variable list (GVL), a program, and a preconfigured generic Modbus TCP slave under the **Modbus TCP IOScanner**. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor the encoder.

In the program, which is part of the Device Module, the read registers of type WORD are mapped into variables of type DWORD.

# Required Libraries

## Required Libraries Used in the Encoder_AbsMlt_ModbusTCP Device Module

The following function block is used in the program organization unit (POU) of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| WORD_AS_DWORD | Toolbox | SE_TBX | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Encoder_AbsMlt_ModbusTCP Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| dwPosition | WORD | Actual encoder position. |
| wPosLow | WORD | Least significant WORD of the present encoder position (DWORD). |
| wPosHigh | WORD | Most significant WORD of the present encoder position (DWORD). |
| diVelocity | DINT | Actual encoder velocity (in points per second (pts/s). |
| wVelLow | WORD | Least significant WORD of the present encoder velocity (DWORD). |
| wVelHigh | WORD | Most significant WORD of the present encoder velocity (DWORD). |

## Program - Prg_<name device module>

### Program Contained in the Encoder_AbsMlt_ModbusTCP Device Module

The program is created in programming language ST (Structured Text).

The program implements the following features:

- Processing of the present position value of the encoder.
- Processing of the present velocity value of the encoder.

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module requires that a **Modbus TCP IOScanner** be added to the Ethernet device network of your controller.

Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can:

- select the **Modbus TCP IOScanner** which manages the device
- assign the IP address for the device

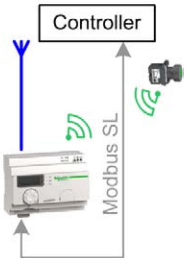
# Section 2.8
## Harmony_Wireless_ModbusSL Device Module

### What Is in This Section?

This section contains the following topics:

## Device Module Description

### Graphical Representation



### Harmony_Wireless_ModbusSL Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to read the signals from a Harmony ZBRN2 wireless receiver. Communication between the SoMachine controller and the receiver is via Modbus SL.

The Device Module Harmony_Wireless_ModbusSL is represented by a function template and consists of a global variable list (GVL), a program, and a preconfigured generic Modbus slave under the **Modbus_IOScanner**. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to apply the wireless receiver in your application. For each channel, a variable of type BOOL is declared and mapped in the I/O mapping of the device. In the program only the monitoring of the communication is processed.

## Required Libraries

### Required Libraries Used in the Harmony_Wireless_ModbusSL Device Module

The following object is used in the program organization units (POU) of the Device Module. The corresponding library is added automatically to the project when the **Modbus_IOScanner** is added.

| Object | Data Type | Library | Namespace | Vendor |
|---|---|---|---|---|
| `MB_CommCode_M238` | ENUM of BYTE | IoDrvModbusSerial | IoDrvModbusSerial | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Harmony_Wireless_ModbusSL Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| `xChannel0…xChannel59` | BOOL | Indicate the state of the channels read from the device. The variables are mapped directly to the inputs in the **I/O mapping** dialog in the device editor. |
| `xComOk` | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE = communication state <> `CommunikationOK` |
| `eComStat` | MB_CommCode_M238 | Communication state of the device. |

## Program - Prg_<name device module>

### Program Contained in the Harmony_Wireless_ModbusSL Device Module

The program is created in programming language CFC (Continuous Function Chart). It provides information on the communication state of the device under the **Modbus_IOScanner**.

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module requires that a **Modbus IOScanner** be added to the serial interface of your controller. Using **Add Function from Template** *(see SoMachine, Programming Guide)*, you can:

- select the **Modbus IOScanner** which manages the device.
- assign the Modbus slave address for the device.

# Section 2.9
## Harmony_Wireless_ModbusTCP_• Device Modules

### Overview

This section provides a generic description for the following Device Modules:

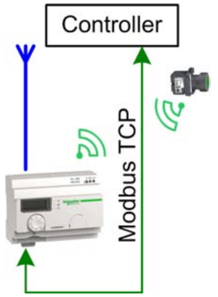
- Harmony_Wireless_ModbusTCP_1
- Harmony_Wireless_ModbusTCP_2

### What Is in This Section?

This section contains the following topics:

## Device Module Description

### Graphical Representation



### Harmony_Wireless_ModbusTCP_• Device Module Description

The Device Modules Harmony_Wireless_ModbusTCP_• provide a ready-to-use coding template as pattern to read the signals from the Harmony ZBRN1 wireless receiver via Modbus TCP through a SoMachine controller.

With the Device Module Harmony_Wireless_ModbusTCP_1, the communication with the wireless receiver is realized in the program code using the corresponding function block. No device is added to your application, but nevertheless the controller must provide an Ethernet interface and the Modbus TCP protocol must be supported.

With the Device Module Harmony_Wireless_ModbusTCP_2, the communication with the wireless receiver is managed by the Modbus TCP IOScanner. The device Harmony ZBRN1 is added to your application; therefore the **Industrial Ethernet manager** is required under the Ethernet interface of the controller.

## Required Libraries

### Required Libraries Used in the Harmony_Wireless_ModbusTCP_• Device Module

The Device Modules implement objects from one or more libraries. The objects and the associated libraries used by these Device Modules are listed in the following table.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| WRITE_VAR | PLCCommunication | SEN | Schneider Electric |
| READ_VAR | | | |
| ADDM | | | |
| IOS_GetHealth | ModbusTCPIOScanner | SEN_IOS | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| OperationErrorCodes | PLCCommunication | SEN | Schneider Electric |
| CommunicationErrorCodes | | | |

## Functional Description

### Device - <name device module>

The Device Module Harmony_Wireless_ModbusTCP_1 does not implement a separate device.

The Device Module Harmony_Wireless_ModbusTCP_2 implements the device Harmony ZBRN1. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** *(see SoMachine, Programming Guide)* dialog box. The device is preconfigured.

The configuration includes the channel for the cyclic data exchange with the device. The repetition rate for the channel is selected with 50 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

The program provided with the Device Module Harmony_Wireless_ModbusTCP_1 contains the program code for the communication with the wireless receiver and the evaluation of the obtained data.

The program provided by the Device Module Harmony_Wireless_ModbusTCP_2 contains only the evaluation of the received data from the wireless receiver because the communication is managed by the **Modbus TCP IOScanner**.

Further information about the program code is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Harmony_Wireless_ModbusTCP_1

The instantiation of this Device Module requires a controller with an Ethernet interface and the Modbus TCP protocol supported. The IP address of the wireless receiver must be provided to the corresponding variable which is used in the program code. This variable is of type **Constant** and its value can be set within the **Add Function From Template** *(see SoMachine, Programming Guide)* dialog box. A later modification of this value is possible too.

Variables selected for IP address parameter:

| Variable | Data type | Default value | Description |
|---|---|---|---|
| `Prg_<module name>.c_sAddr` | STRING | 3{0.0.0.0} | IP Address (RFID smart antenna) configuration used by the communication function blocks in the program.<br>Format: '`<communication link>{<IP address A.B.C.D>:<port>}<UnitID>`'<br>**NOTE:** If the `<port>` is not included in the string, the default '502' is used. |

## Instantiation of the Harmony_Wireless_ModbusTCP_2

The instantiation of this Device Module requires the **Industrial Ethernet manager** under the Ethernet interface of your controller.

Using **Add Function From Template**, you can:

- Select the fieldbus master which manages the device
- Assign the IP address for the device

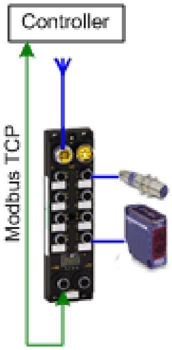
# Section 2.10
## IO_ETB_ModbusTCP Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## IO_ETB_ModbusTCP Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control an Advantys ETB I/O block via Modbus TCP through a SoMachine controller.

The Device Module IO_ETB_ModbusTCP is represented by a function template and consists of a global variable list (GVL) and a preconfigured generic Modbus TCP slave under the **Modbus TCP IOScanner**. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the IO_ETB_ModbusTCP Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xWritePoint1A | BOOL | Set output signal of pin 1. A. |
| xWritePoint1B | BOOL | Set output signal of pin 1.B. |
| ... | ... | ... |
| xWritePoint8A | BOOL | Set output signal of pin 8. A. |
| xWritePoint8B | BOOL | Set output signal of pin 8.B. |
| xReadPoint1A | BOOL | Read status of pin 1. A. |
| xReadPoint1B | BOOL | Read status of pin 1.B. |
| xReadPoint8A | BOOL | Read status of pin 8. A. |
| xReadPoint8B | BOOL | Read status of pin 8.B. |
| wWatchdogValue | WORD | Watchdog value<br>This value is multiplied by a factor of 100 to produce the watchdog timeout setting. Applied to points with fallback set to ON. |
| wWatchdogState | WORD | Watchdog state<br>0 = not active<br>1 = active |
| wWatchdogBehavior | WORD | 0 = apply output fallback<br>1 = hold output |
| wFirmwareState | WORD | 0 = OK<br>1 = error detected on firmware |
| wFallbackForPoint1A | WORD | Pin 1. A<br>0 = OFF<br>Value unequal to 0 = ON (used when I/O pin is an output) |
| wFallbackForPoint1B | WORD | Pin 1.B<br>0 = OFF<br>Value unequal to 0 = ON (used when I/O pin is an output) |
| ... | ... | ... |
| wFallbackForPoint8A | WORD | Pin 8. A<br>0 = OFF<br>Value unequal to 0 = ON (used when I/O pin is an output) |
| wFallbackForPoint8B | WORD | Pin 8.B<br>0 = OFF<br>Value unequal to 0 = ON (used when I/O pin is an output) |

| Variable | Data Type | Description |
|---|---|---|
| `wIOconfigPoint1A` | WORD | I/O configuration for pin 1.A:<br>0 = input pin<br>1 = output pin<br>2 = universal I/O |
| `wIOconfigPoint1B` | WORD | I/O configuration for pin 1.B:<br>0 = input pin<br>1 = output pin<br>2 = universal I/O |
| ... | ... | ... |
| `wIOconfigPoint8A` | WORD | I/O configuration for pin 8.A:<br>0 = input pin<br>1 = output pin<br>2 = universal I/O |
| `wIOconfigPoint8B` | WORD | I/O configuration for pin 8.B:<br>0 = input pin<br>1 = output pin<br>2 = universal I/O |

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires that a serial line interface configured as **Modbus TCP IOScanner** be added to the Ethernet device network of your controller. Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can assign the Modbus TCP address for the device.

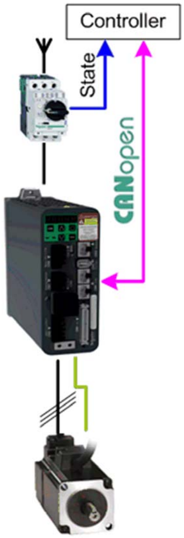
# Section 2.11
## Lexium_28_CANopen Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_28_CANopen Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium 28 servo drive via CANopen through a SoMachine controller.

The Device Module Lexium_28_CANopen is represented by a function template and consists of a global variable list (GVL), a program, and the device Lexium 28 under the CANopen manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the Lexium 28 via CANopen.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device in jog mode
- control the device in velocity mode
- control the device in relative positioning mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

With the Lexium 28 as part of this Device Module, the second and the third transmit PDOs (Process Data Object) are activated to monitor the values for speed and position of the drive.

In addition, for the transmit PDOs the inhibit time is set to 10 ms and the event time is set to 100 ms. Therefore, there is a transmission between 10 ms (inhibit time) given data in the PDO has changed, and 100 ms (event time) given no data has changed.

**NOTE:** If you do not need this feature for your application, unselect the second and third transmit PDO (Process Data Object) on the tab **PDO mapping** in the device editor of the drive to optimize your data transmission rate.

# Required Libraries

## Required Libraries Used in the Lexium_28_CANopen Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power_LXM28 | Lexium 28 Library | SEM_LXM28 | Schneider Electric |
| MC_Reset_LXM28 | | | |
| MC_Stop_LXM28 | | | |
| MC_Jog_LXM28 | | | |
| MC_MoveVelocity_LXM28 | | | |
| MC_MoveRelative_LXM28 | | | |
| MC_MoveAbsolute_LXM28 | | | |
| MC_ReadAxisError_LXM28 | | | |
| MC_Home_LXM28 | | | |
| GET_STATE | CAA CiA 405 | CIA405 | CAA Technical Workgroup |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Lexium_28_CANopen Device Module

The table presents the variables provided with the global variable list:

| Variable | Data type | Description |
|---|---|---|
| `xMcbRdy` | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| `diActlVelo` | DINT | Indicates the velocity of the drive in rpm. This variable is already mapped to the object **Velocity actual value** in the tab **CANopen I/O mapping** of the device editor. |
| `diActPos` | DINT | Indicates the position of the drive. This variable is already mapped to the object **Positon actual value** in the tab **CANopen I/O mapping** of the device editor. |
| `xCmdEnPwr` | BOOL | Enables the power stage of the drive. |
| `xCmdRst` | BOOL | Resets the drive in case of an error state. |
| `xCmdStop` | BOOL | Stops the drive. |
| `xCmdJogFwd` | BOOL | Jogs the drive in a forward direction. |
| `xCmdJogRev` | BOOL | Jogs the drive in a reverse direction. |
| `xCmdJogFast` | BOOL | Defines the velocity setpoint (fast or slow) for jog operation. |
| `diSetJogDist` | DINT | Defines the distance to move for one interval on jog operation. If the value is set to 0, continuous motion is used. |
| `iWaitTimeJog` | INT | Defines the time delay in ms for change to continuous motion during jog operation via distance. |
| `diSetVeloJogSlow` | DINT | Velocity setpoint for jog operation at slow speed. |
| `diSetVeloJogFast` | DINT | Velocity setpoint for jog operation at fast speed. |
| `xCmdMovVelo` | BOOL | Starts the drive with continuous velocity |
| `xCmdMovRel` | BOOL | Starts the drive for relative positioning |
| `xCmdMovAbs` | BOOL | Starts the drive for absolute positioning |
| `diSetVeloMovVelo` | DINT | Velocity setpoint for velocity mode in rpm. |
| `diSetVeloMovRel` | DINT | Velocity setpoint for relative positioning in rpm. |
| `diSetVeloMovAbs` | DINT | Velocity setpoint for absolute positioning in rpm. |
| `diSetDistMovRel` | DINT | Distance for relative positioning in increments. |
| `diSetPosMovAbs` | DINT | Target position for absolute positioning in increments. |
| `xCmdHoming` | BOOL | Starts homing operation. |
| `diSetHomePos` | DINT | Position to set if homing is finished. |
| `uiSetHomeMod` | UINT | Defines the method for homing operation. |

| Variable | Data type | Description |
|---|---|---|
| diSetVeloHome | DINT | Velocity setpoint for search of the reference switch in rpm. |
| diSetVeloOutHome | DINT | Velocity setpoint for movement back to edge of reference switch in rpm. |
| xStatEnbl | BOOL | Indicates the state of the power stage. |
| wErrID | WORD | Indicates the error ID of the detected error. Refer to the *Lexium 28 Library Function Blocks Software Manual*. |
| xErr | BOOL | Indicates that an error state exists. |
| xVeloActv | BOOL | Indicates that the continuous velocity operation is active. |
| xRelActv | BOOL | Indicates that the relative positioning operation is active. |
| xAbsActv | BOOL | Indicates that the absolute positioning operation is active. |
| xHomeActv | BOOL | Indicates that the homing operation is active. |
| xJogActv | BOOL | Indicates that the jog operation is active. |
| xComOk | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| eComSta | CIA405.DEVICE_STATE | Communication state of the device. For information on the enumeration, refer to the CIA405 Library Guide (see SoMachine Online Help under *CoDeSys Libraries/CAA Libraries/CAA_CiA405.library*). |

## Program - Prg_<module name>

### Program Contained in the Lexium_28_CANopen Device Module

The program is divided into two actions and is created in programming language CFC (Continuous Function Chart). Both actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_Lexium

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANopen fieldbus. Based on the communication parameter, the communication state is assigned to the corresponding variables which have been declared in the `GVL_<module name>`. If the state is equal to `Operational`, the variable for the state indicates TRUE and other cases are indicated by FALSE.

### Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power_LXM28`, enable/disable the power stage of the drive
- with the function block `MC_Reset_LXM28`, reset the drive after an error
- with the function block `MC_Stop_LXM28`, stop operation on the drive
- with the function block `MC_Jog_LXM28`, operate the drive in jog mode
- with the function block `MC_MoveVelocity_LXM28`, operate the drive with continuous velocity
- with the function block `MC_MoveRelative_LXM28`, operate the drive with relative positioning
- with the function block `MC_MoveAbsolute_LXM28`, operate the drive with absolute positioning
- with the function block `MC_ReadAxisError_LXM28`, obtain the error state of the drive
- with the function block `MC_Home_LXM28`, initiate the homing mode

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_CANopen. For more information, refer to Adding Device Module to the Project *(see page 47)*.

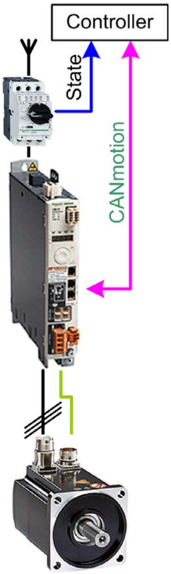
# Section 2.12
## Lexium_32A_CANmotion Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_32A_CANmotion Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium 32A via CANmotion through a SoMachine controller.

The Device Module Lexium_32A_CANmotion is represented by a function template and consists of a global variable list, a program, and the device Lexium 32A under the CANmotion manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the Lexium 32A via CANmotion.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device in velocity mode
- control the device in relative positioning mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

# Required Libraries

## Required Libraries Used in the Lexium_32A_CANmotion Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power | SM3_Basic | SM3_Basic | 3S - Smart Software Solutions GmbH |
| MC_Reset | | | |
| MC_Stop | | | |
| MC_MoveVelocity | | | |
| MC_MoveRelative | | | |
| MC_MoveAbsolute | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadActualPosition | | | |
| MC_ReadAxisError | | | |
| MC_ReadParameter | | | |
| SMC3_ReinitDrive | | | |

## Global Variable List - GVL_<name device module>

### Global Variables Provided in the Lexium_32A_CANmotion Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xMcbRdy | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| xCmdEnPwr | BOOL | Enables power to the drive. |
| xQuickStop | BOOL | Disables the quick stop mechanism. |
| xCmdRst | BOOL | Resets an error state on the axis. |
| xCmdReinitDrive | BOOL | Reinitializes the axis (start-up phase is reactivated). |
| xCmdStop | BOOL | Stops the axis. |
| xCmdHoming | BOOL | Starts homing operation. |
| xCmdMovVelo | BOOL | Starts the axis with continuous velocity. |
| xCmdMovAbs | BOOL | Starts the axis with absolute positioning. |
| xCmdMovRel | BOOL | Starts the axis with relative positioning. |
| lrSetHomePos | LREAL | Position to set if homing is finished. |
| lrSetVeloMovVelo | LREAL | Velocity setpoint for velocity mode in u/s. |
| lrSetVeloMovAbs | LREAL | Velocity setpoint for absolute positioning u/s. |
| lrSetPosMovAbs | LREAL | Target position for absolute positioning in technical units. |
| lrSetVeloMovRel | LREAL | Velocity setpoint for relative positioning u/s. |
| lrSetDistMovRel | LREAL | Distance for relative positioning in technical units. |
| lrSetAcc | LREAL | Value of the acceleration [u/s$^2$]. |
| lrSetDec | LREAL | Value of the deceleration [u/s$^2$]. |
| eDirMovVelo | SM3_Basic.MC_DIRECTION | Direction for continuous velocity operation:<br>-1 = negative<br>1 = positive<br>2 = the active direction |
| xStatEnbl | BOOL | Indicates whether the drive is powered and quick stop mechanism is disabled. |
| xHomeActv | BOOL | Indicates that the homing operation is active. |
| xAbsActv | BOOL | Indicates that the absolute positioning operation is active. |
| xVeloActv | BOOL | Indicates that the continuous velocity operation is active. |

| Variable | Data Type | Description |
|---|---|---|
| xRelActv | BOOL | Indicates that the relative positioning operation is active. |
| xErr | BOOL | Indicates that an error state exists. |
| xActlPosVld | BOOL | Indicates whether the value lrActlPos is valid. |
| lrActlPos | LREAL | Position of the axis unit [u]. |
| xActlVeloVld | BOOL | Indicates whether the value lrActlVelo is valid. |
| lrActlVelo | LREAL | Velocity of axis unit [u/s]. |
| xErrIdVld | BOOL | Indicates whether the value dwErrId is valid. |
| dwErrId | DWORD | Vendor-specific value of the axis error. |
| xAxisStatVld | BOOL | Indicates whether the value eAxisStat is valid. |
| eAxisStat | SM3_Basic.SMC_AXIS_STATE | State of the axis according to PLCopen state diagram. |
| xComOk | BOOL | Indicates the CANmotion communication state.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| diHmiSetHomePos | DINT | Position to set if homing is finished. |
| diHmiSetVeloMovVelo | DINT | Velocity setpoint for velocity mode in u/s. |
| diHmiSetVeloMovAbs | DINT | Velocity setpoint for absolute positioning u/s. |
| diHmiSetPosMovAbs | DINT | Target position for absolute positioning in technical units. |
| diHmiSetVeloMovRel | DINT | Velocity setpoint for relative positioning u/s. |
| diHmiSetDistMovRel | DINT | Distance for relative positioning in technical. units. |
| diHmiActlPos | DINT | Position of axis unit [u]. |
| diHmiActlVelo | DINT | Velocity of axis unit [u/s]. |
| diHmiAcc | DINT | Value of the acceleration [u/s$^2$]. |
| diHmiDec | DINT | Value of the deceleration [u/s$^2$]. |

## Program - Prg_<name device module>

### Program Contained in the Lexium_32A_CANmotion Device Module

The program is divided into 3 actions and is created in programming language CFC (Continuous Function Chart). These actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_Lexium
- Action - A03_HmiVarConversion

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANmotion bus. The communication state is provided by an element of the axis structure and is assigned to the corresponding variable which have been declared in the `GVL_<module name>`.

If the CANmotion communication is OK, the variable for the general state will indicate TRUE, otherwise the state of the variable is FALSE.

### Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power`, enable/disable the power stage of the drive
- with the function block `MC_Reset`, reset the drive after an error
- with the function block `SMC3_ReinitDrive`, reinitialize the axis
- with the function block `MC_Stop`, stop operation on the drive
- with the function block `MC_Home`, initiate the homing mode
- with the function block `MC_MoveVelocity`, operate the drive with continuous velocity
- with the function block `MC_MoveAbsolute`, operate the drive with absolute positioning
- with the function block `MC_MoveRelative`, operate the drive with relative positioning
- with the function block `MC_ReadActualVelocity`, read the velocity
- with the function block `MC_ReadActualPosition`, read the position
- with the function block `MC_ReadAxisError`, obtain the error state of the drive
- with the function block `MC_ReadParameter`, obtain the status of the drive

### Action - A03_HmiVarConversion

Not all HMI devices support the datatype LREAL, therefore variables of the datatype DINT have been declared with the same meaning. In this action the HMI variables of type DINT will be converted and assigned to the process variables of type LREAL.

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module requires that a CANmotion manager be added to the CAN interface of your controller.

Using the instantiation, you can:

- select the CANmotion manager which shall manage the device
- assign the CAN node address for the device
- map variables to physical inputs and outputs of your configuration

Variable selected for I/O mapping (input/output)

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdy` | BOOL | – | Indicates the state of Motor Circuit Breaker |

**NOTE:** When the Device Module has been added to your application, you must configure correctly the **SoftMotion** drive (`SM_<module name>`). The drive settings have to be adapted according to your hardware and the application.

⚠ **WARNING**

**UNINTENDED EQUIPMENT OPERATION**

Configure the **SoftMotion** drive by editing the default parameters to those that conform to both your hardware and application needs.

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

Double-click the **SM drive** (`SM_<module name>`) to open the appropriate tab.

The tab **SM drive** includes the following tabs:

- **SoftMotion Drive: Basic**
  Here you can make the settings for axis type, limits, and velocity ramp type.
- **SoftMotion Drive: Scaling/Mapping**
  Here you can define the scaling between motor encoder increments and units in application.

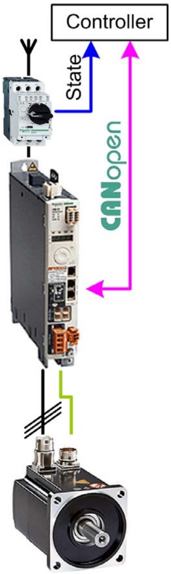
# Section 2.13
## Lexium_32A_CANopen Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_32A_CANopen Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium 32A via CANopen through a SoMachine controller.

The Device Module Lexium_32A_CANopen is represented by a function template and consists of a global variable list (GVL), a program, and the device Lexium 32A under the CANopen manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the Lexium 32A via CANopen.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device in jog mode
- control the device in velocity mode
- control the device in relative positioning mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

With the Lexium 32A as part of this Device Module, the second and the third transmit PDO (Process Data Object) are activated to monitor the values for speed and position of the drive.

In addition, for the transmit PDOs, the inhibit time is set to 10 ms. Therefore, given data in the PDO is constantly changing, there is a transmission every 10 ms.

**NOTE:** If you do not need this feature for your application, unselect the second and third transmit PDO (Process Data Object) on the tab **PDO mapping** of the drive to optimize your data transmission rate.

# Required Libraries

## Required Libraries Used in the Lexium_32A_CANopen Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power_LXM | Lexium Library | SEM_LXM | Schneider Electric |
| MC_Reset_LXM | | | |
| MC_Stop_LXM | | | |
| MC_Jog_LXM | | | |
| MC_MoveVelocity_LXM | | | |
| MC_MoveRelative LXM | | | |
| MC_MoveAbsolute_LXM | | | |
| MC_ReadAxisError_LXM | | | |
| MC_Home_LXM | | | |
| GET_STATE | CAA CiA 405 | CIA405 | CAA Technical Workgroup |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Lexium_32A_CANopen Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| `xMcbRdy` | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| `diActlVelo` | DINT | Indicates the velocity of the drive in rpm. This variable is already mapped to the object **Velocity actual value** in the device dialog **CANopen I/O mapping**. |
| `diActPos` | DINT | Indicates the position of the drive. This variable is already mapped to the object **Positon actual value** in the device dialog **CANopen I/O mapping**. |
| `xCmdEnPwr` | BOOL | Enables the power stage of the drive. |
| `xCmdRst` | BOOL | Resets the drive in case of an error state. |
| `xCmdStop` | BOOL | Stops the drive. |
| `xCmdJogFwd` | BOOL | Jogs the drive in a forward direction. |
| `xCmdJogRev` | BOOL | Jogs the drive in a reverse direction. |
| `xCmdJogFast` | BOOL | Defines the velocity setpoint (fast or slow) for jog operation. |
| `diSetJogDist` | DINT | Defines the distance to move for one interval on jog operation. If the value is set to 0, continuous motion is used. |
| `iWaitTimeJog` | INT | Defines the time delay in ms for change to continuous motion. |
| `diSetVeloJogSlow` | DINT | Velocity setpoint for jog operation at slow speed. |
| `diSetVeloJogFast` | DINT | Velocity setpoint for jog operation at fast speed. |
| `xCmdMovVelo` | BOOL | Starts the drive with continuous velocity |
| `xCmdMovRel` | BOOL | Starts the drive for relative positioning |
| `xCmdMovAbs` | BOOL | Starts the drive for absolute positioning |
| `diSetVeloMovVelo` | DINT | Velocity setpoint for velocity mode in rpm. |
| `diSetVeloMovRel` | DINT | Velocity setpoint for relative positioning in rpm. |
| `diSetVeloMovAbs` | DINT | Velocity setpoint for absolute positioning in rpm. |
| `diSetDistMovRel` | DINT | Distance for relative positioning in increments. |
| `diSetPosMovAbs` | DINT | Target position for absolute positioning in increments. |
| `xCmdHoming` | BOOL | Starts homing operation. |
| `diSetHomePos` | DINT | Position to set if homing is finished. |
| `uiSetHomeMod` | UINT | Defines the method for homing operation. |
| `diSetVeloHome` | DINT | Velocity setpoint for search of the reference switch. |

| Variable | Data Type | Description |
|---|---|---|
| `diSetVeloOutHome` | DINT | Velocity setpoint for movement back to edge of reference switch. |
| `diSetPosOutHome` | DINT | Maximum distance for movement back to edge of reference switch. |
| `diSetPosDisHome` | DINT | Distance for positioning starting from edge of reference switch. |
| `xStatEnbl` | BOOL | Indicates the state of the power stage. |
| `wErrID` | WORD | Indicates the error ID of the detected error. Refer to the *Lexium Library Function Blocks Software Manual*. |
| `xErr` | BOOL | Indicates that an error state exists. |
| `xVeloActv` | BOOL | Indicates that the continuous velocity operation is active. |
| `xRelActv` | BOOL | Indicates that the relative positioning operation is active. |
| `xAbsActv` | BOOL | Indicates that the absolute positioning operation is active. |
| `xHomeActv` | BOOL | Indicates that the homing operation is active. |
| `xJogActv` | BOOL | Indicates that the jogging operation is active. |
| `xComOk` | BOOL | Indicates the CANmotion communication state.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| `eComStat` | CIA405.DEVICE_STATE | Communication state of the device. For information on the enumeration, refer to the CIA405 Library Guide (see SoMachine Online Help under *CoDeSys Libraries/CAA Libraries/CAA_CiA405.library*). |

## Program - Prg_<name device module>

### Program Contained in the Lexium_32A_CANopen Device Module

The program is divided into 2 actions and is created in programming language CFC (Continuous Function Chart). Both actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_Lexium

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANopen fieldbus. Based on the communication parameter the communication state is assigned to the corresponding variables which have been declared in the GVL_<module name>. If the state is equal to `OPERATIONAL`, the variable for the state indicates TRUE and other cases are indicated by FALSE.

### Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power_LXM`, enable/disable the power stage of the drive
- with the function block `MC_Reset_LXM`, reset the drive after an error
- with the function block `MC_Stop_LXM`, stop operation on the drive
- with the function block `MC_Jog_LXM`, operate the drive in jog mode
- with the function block `MC_MoveVelocity_LXM`, operate the drive with continuous velocity
- with the function block `MC_MoveRelative_LXM`, operate the drive with relative positioning
- with the function block `MC_MoveAbsolute_LXM`, operate the drive with absolute positioning
- with the function block `MC_ReadAxisError_LXM`, obtain the error state of the drive
- with the function block `MC_Home_LXM`, initiate the homing mode

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_CANopen. For more information, refer to Adding Device Module to the Project *(see page 47)*.

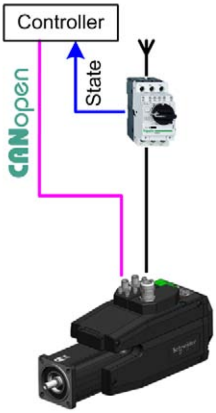
# Section 2.14
# Lexium_32i_CANopen Device Module

## What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_32i_CANopen Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium 32i via CANopen through a SoMachine controller.

The Device Module Lexium_32i_CANopen is represented by a function template and consists of a global variable list (GVL), a program, and the device Lexium 32i under the CANopen manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the Lexium 32i via CANopen.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device in jog mode
- control the device in velocity mode
- control the device in relative positioning mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

With the Lexium 32i as part of this Device Module, the second and the third transmit PDOs (Process Data Object) are activated to monitor the values for speed and position of the drive.

In addition, for the transmit PDOs, the inhibit time is set to 10 ms. Therefore, given data in the PDO is constantly changing, there is a transmission every 10 ms.

**NOTE:** If you do not need this feature for your application, unselect the second and third transmit PDO on the tab **PDO mapping** of the drive to optimize your data transmission rate.

# Required Libraries

## Required Libraries Used in the Lexium_32i_CANopen Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power_LXM32i | Lexium 32i Library | SEM_LXM32i | Schneider Electric |
| MC_Reset_LXM32i | | | |
| MC_Stop_LXM32i | | | |
| MC_Jog_LXM32i | | | |
| MC_MoveVelocity_LXM32i | | | |
| MC_MoveRelative LXM32i | | | |
| MC_MoveAbsolute_LXM32i | | | |
| MC_ReadAxisError_LXM32i | | | |
| MC_Home_LXM32i | | | |
| GET_STATE | CAA CiA 405 | CIA405 | CAA Technical Workgroup |

## Global Variable List - GVL_<name device module>

### Global Variables Provided by the Lexium_32i_CANopen Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| `xMcbRdy` | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| `diActlVelo` | DINT | Indicates the velocity of the drive in rpm. This variable is already mapped to the object **Velocity actual value** in the device dialog **CANopen I/O mapping**. |
| `diActPos` | DINT | Indicates the position of the drive in rpm. This variable is already mapped to the object **Positon actual value** in the device dialog **CANopen I/O mapping**. |
| `xCmdEnPwr` | BOOL | Enables the power stage of the drive. |
| `xCmdRst` | BOOL | Resets the drive in case of an error state. |
| `xCmdStop` | BOOL | Stops the drive. |
| `xCmdJogFwd` | BOOL | Jogs the drive in a forward direction. |
| `xCmdJogRev` | BOOL | Jogs the drive in a reverse direction. |
| `xCmdJogFast` | BOOL | Defines the velocity setpoint (fast or slow) for jog operation. |
| `diSetJogDist` | DINT | Defines the distance to move for one interval on jog operation. If the value is set to 0, continuous motion is used. |
| `iWaitTimeJog` | INT | Defines the time delay in ms for change to continuous motion. |
| `diSetVeloJogSlow` | DINT | Velocity setpoint for jog operation at slow speed. |
| `diSetVeloJogFast` | DINT | Velocity setpoint for jog operation at fast speed. |
| `xCmdMovVelo` | BOOL | Starts the drive with continuous velocity |
| `xCmdMovRel` | BOOL | Starts the drive for relative positioning |
| `xCmdMovAbs` | BOOL | Starts the drive for absolute positioning |
| `diSetVeloMovVelo` | DINT | Velocity setpoint for velocity mode in rpm. |
| `diSetVeloMovRel` | DINT | Velocity setpoint for relative positioning in rpm. |
| `diSetVeloMovAbs` | DINT | Velocity setpoint for absolute positioning in rpm. |
| `diSetDistMovRel` | DINT | Distance for relative positioning in increments. |
| `diSetPosMovAbs` | DINT | Target position for absolute positioning in increments. |
| `xCmdHoming` | BOOL | Starts homing operation. |
| `diSetHomePos` | DINT | Position to set if homing is finished. |

| Variable | Data Type | Description |
|---|---|---|
| `uiSetHomeMod` | UINT | Defines the method for homing operation. |
| `diSetVeloHome` | DINT | Velocity setpoint for search of the reference switch. |
| `diSetVeloOutHome` | DINT | Velocity setpoint for movement back to edge of reference switch. |
| `diSetPosOutHome` | DINT | Maximum distance for movement back to edge of reference switch. |
| `diSetPosDisHome` | DINT | Distance for positioning starting from edge of reference switch. |
| `diSetDecStop` | DINT | Deceleration setpoint for stopping an operation. |
| `xStatEnbl` | BOOL | Indicates the state of the power stage. |
| `wErrID` | WORD | Indicates the error ID of the detected error. Refer to the *Lexium Library Function Blocks Software Manual*. |
| `xErr` | BOOL | Indicates that an error state exists. |
| `xVeloActv` | BOOL | Indicates that the continuous velocity operation is active. |
| `xRelActv` | BOOL | Indicates that the relative positioning operation is active. |
| `xAbsActv` | BOOL | Indicates that the absolute positioning operation is active. |
| `xHomeActv` | BOOL | Indicates that the homing operation is active. |
| `xJogActv` | BOOL | Indicates that the jogging operation is active. |
| `xComOk` | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| `eComStat` | CIA405.DEVICE_STATE | Communication state of the device. Enumeration, refer to the CIA405 Library Guide (see SoMachine Online Help under *CoDeSys Libraries/CAA Libraries/CAA_CiA405.library*). |

## Program - Prg_<name device module>

### Program Contained in the Lexium_32i_CANopen Device Module

The program is divided into 2 actions and is created in programming language CFC (Continuous Function Chart). Both actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_Lexium

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANopen fieldbus. Based on the communication parameter the communication state is assigned to the corresponding variables which have been declared in the GVL_<module name>. If the state is equal to `OPERATIONAL`, the variable for the state indicates TRUE and other cases are indicated by FALSE.

### Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power_LXM32i`, enable/disable the power stage of the drive
- with the function block `MC_Reset_LXM32i`, reset the drive after an error
- with the function block `MC_Stop_LXM32i`, stop operation on the drive
- with the function block `MC_Jog_LXM32i`, operate the drive in jog mode
- with the function block `MC_MoveVelocity_LXM32i`, operate the drive with continuous velocity
- with the function block `MC_MoveRelative_LXM32i`, operate the drive with relative positioning
- with the function block `MC_MoveAbsolute_LXM32i`, operate the drive with absolute positioning
- with the function block `MC_ReadAxisError_LXM32i`, obtain the error state of the drive
- with the function block `MC_Home_LXM32i`, initiate the homing mode

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_CANopen. For more information, refer to Adding Device Module to the Project *(see page 47)*.

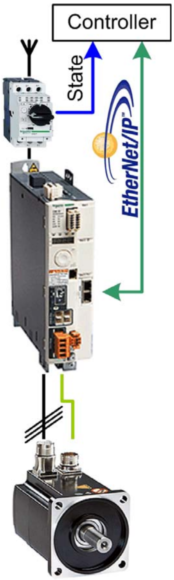
# Section 2.15
# Lexium_32M_EtherNetIP Device Module

## What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_32M_EtherNetIP Device Module Description

The Device Module Lexium_32M_EtherNetIP provides the application objects and the device which are required to monitor and control a Lexium 32M via EtherNet/IP with a Schneider Electric SoMachine controller. The device Lexium 32M requires the **Industrial Ethernet manager** under the Ethernet interface of the controller.

# Required Libraries

## Required Libraries Used in the Lexium_32M_EtherNetIP Device Module

The Device Module implements objects from one or more libraries. The objects and the associated libraries used by this Device Module are listed in the following tables.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power | GMC Independent PLCopen MC | GIPLC | Schneider Electric |
| MC_Reset | | | |
| MC_MoveVelocity | | | |
| MC_MoveRelative | | | |
| MC_MoveAbsolute | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadActualPosition | | | |
| MC_ReadStatus | | | |
| MC_ReadAxisInfo | | | |
| MC_ReadAxisError | | | |
| Stop_LXM32 | GMC Independent Lexium | GILXM | |
| Halt_LXM32 | | | |
| Home_LXM32 | | | |
| Jog_LXM32 | | | |
| SetLimitSwitch_LXM32 | | | |
| SetDriveRamp_LXM32 | | | |
| SetStopRamp_LXM32 | | | |
| StoreParameters_LXM32 | | | |
| EIPGetHealthBit | EtherNetIP Scanner | EIPSC | |
| EIPStartConnection | | | |
| EIPStopConnection | | | |
| FB_RemoteAdapter | EtherNetIP Remote Adapter | EIPRA | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| `OperationErrorCodes` | EtherNetIP Scanner | EIPSC | Schneider Electric |
| `CIPOperationErrorCodes` | | | |
| `CommunicationErrorCodes` | | | |
| `eStatus` | EtherNetIP Remote Adapter | EIPRA | |
| `eAdapterErrorInfo` | | | |

**NOTE:** The library EtherNetIP Scanner is not supported by the motion controller LMC078.

## Functional Description

### Device - <name device module>

The Device Module implements the device Lexium 32M for EtherNet/IP. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** dialog box.

The device is preconfigured. The configuration includes the connection for the drive profile Lexium 32M with the assemblies 103 (output) and 113 (input). The Request Packet Interval (RPI) is selected with 10 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

For basic control functions, the program code does not need to be modified, all required signals and parameter are linked to the associated variables in the GVL.

The program is divided into several actions. These are described in the following table.

**NOTE:** The program logic of the action `A01_ComCtrl` is not supported in an application of a motion controller LMC078.

| Name of the action | Description |
|---|---|
| `A01_ComCtrl` | Processes the functions to monitoring and control of the EtherNet/IP communication with the device. |
| `A02_Ctrl_LXM` | Contains a selection of function block calls to control the Lexium. Each function block is called in each program cycle. |
| `A03_Stat_LXM` | Contains a selection of function block calls to gather status information from the Lexium. Each function block is called in each program cycle. |
| `A04_Config_LXM` | Contains a selection of function block calls to write a set of parameters to the Lexium. |

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_EtherNetIP. For more information, refer to Adding Device Module to the Project *(see page 33)*.

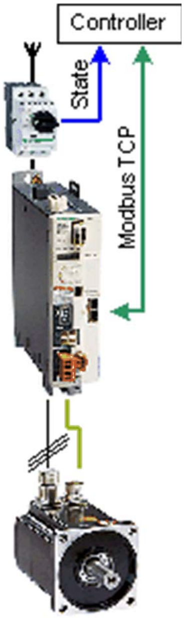
# Section 2.16
## Lexium_32M_ModbusTCP Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_32M_ModbusTCP Device Module Description

The Device Module Lexium_32M_ModbusTCP provides the application objects and the device which are required to monitor and control a Lexium 32M via Modbus TCP with a Schneider Electric SoMachine controller. The device Lexium 32M requires the **Industrial Ethernet manager** under the Ethernet interface of the controller.

# Required Libraries

## Required Libraries Used in the Lexium_32M_ModbusTCP Device Module

The Device Module implements objects from one or more libraries. The objects and the associated libraries used by this Device Module are listed in the following table.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power | GMC Independent PLCopen MC | GIPLC | Schneider Electric |
| MC_Reset | | | |
| MC_MoveVelocity | | | |
| MC_MoveRelative | | | |
| MC_MoveAbsolute | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadActualPosition | | | |
| MC_ReadStatus | | | |
| MC_ReadAxisInfo | | | |
| MC_ReadAxisError | | | |
| Stop_LXM32 | GMC Independent Lexium | GILXM | |
| HALT_LXM32 | | | |
| Home_LXM32 | | | |
| Jog_LXM32 | | | |
| SetLimitSwitch_LXM32 | | | |
| SetDriveRamp_LXM32 | | | |
| SetStopRamp_LXM32 | | | |
| StoreParameters_LXM32 | | | |
| IOS_GetHealth | ModbusTCPIOScanner | SE_IOS | |

## Functional Description

### Device - <name device module>

The Device Module implements the device Lexium 32M for Modbus TCP. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** *(see SoMachine, Programming Guide)* dialog box.

The device is preconfigured. The configuration includes the Lexium 32M channel for the cyclic data exchange with the device. The repetition rate for the channel is selected with 10 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

For basic control functions, the program code does not need to be modified, all required signals and parameter are linked to the associated variables in the GVL.

The program is divided into several actions. These are described in the following table.

| Name of the action | Description |
|---|---|
| `A01_ComStat` | Processes the functions to monitoring and control of the Modbus TCP communication with the device. |
| `A02_Ctrl_LXM` | Contains a selection of function block calls to control the Lexium. Each function block is called in each program cycle. |
| `A03_Stat_LXM` | Contains a selection of function block calls to gather status information from the Lexium. Each function block is called in each program cycle. |
| `A04_Config_LXM` | Contains a selection of function block calls to write a set of parameters to the Lexium. |

**NOTE:** For monitoring the communication state of the device the channel ID of the configured Modbus TCP channel must be set as value for the variable `GVL_<name device module>.c_uiChannelId`. The channel ID is automatically generated when the device is added to the project and can be obtained through the **Device Editor** in the tab **Modbus TCP Channel configuration**.

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_ModbusTCP. For more information, refer to Adding Device Module to the Project *(see page 33)*.

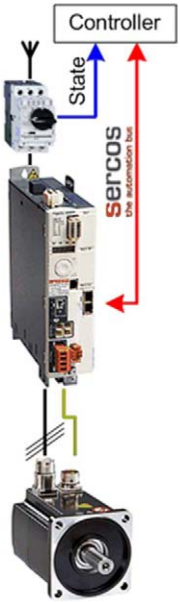
# Section 2.17
## Lexium_32S_Sercos Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_32S_Sercos Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium 32S via Sercos motion bus through a SoMachine controller.

The Device Module Lexium_32S_Sercos is represented by a function template and consists of a global variable list (GVL), a program, and the device Lexium 32S under the Sercos interface of the controller. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the servo drive Lexium 32S via Sercos motion bus.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device in velocity mode
- control the device in relative positioning mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

# Required Libraries

## Required Libraries Used in the Lexium_32S_Sercos Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function block | Library | Namespace | Vendor |
|---|---|---|---|
| `MC_Power` | SM3_Basic | SM3_Basic | 3S - Smart Software Solutions GmbH |
| `MC_Reset` | | | |
| `MC_Stop` | | | |
| `MC_Home` | | | |
| `MC_MoveVelocity` | | | |
| `MC_MoveRelative` | | | |
| `MC_MoveAbsolute` | | | |
| `MC_ReadActualVelocity` | | | |
| `MC_ReadActualPosition` | | | |
| `MC_ReadAxisError` | | | |
| `MC_ReadParameter` | | | |
| `SMC3_ReinitAxis` | | | |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Lexium_32S_Sercos Device Module

The table presents the variables provided with the global variable list:

| Variable | Data type | Description |
|---|---|---|
| `xEnable` | BOOL | Signal enables the execution of the POU which belongs to this Device Module. |
| `xMcbRdy` | BOOL | Indicates the state of Motor Circuit Breaker. Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| `xCmdEnPwr` | BOOL | Enables the power stage of the drive. |
| `xQuickStop` | BOOL | Signal disables the quick stop mechanism. |
| `xCmdRst` | BOOL | Resets an error state on the axis. |
| `xCmdReinitDrive` | BOOL | Reinitializes the axis (start-up phase is reactivated). |
| `xCmdStop` | BOOL | Stops the axis. |
| `xCmdHoming` | BOOL | Starts homing operation. |
| `xCmdMovVelo` | BOOL | Starts the axis with continuous velocity. |
| `xCmdMovAbs` | BOOL | Starts the axis with absolute positioning. |
| `xCmdMovRel` | BOOL | Starts the axis with relative positioning. |
| `lrSetHomePos` | LREAL | Position to set if homing is finished. |
| `lrSetVeloMovVelo` | LREAL | Velocity setpoint for velocity mode in u/s. |
| `lrSetVeloMovAbs` | LREAL | Velocity setpoint for absolute positioning u/s. |
| `lrSetPosMovAbs` | LREAL | Target position for absolute positioning in technical units. |
| `lrSetVeloMovRel` | LREAL | Velocity setpoint for relative positioning u/s. |
| `lrSetDistMovRel` | LREAL | Distance for relative positioning in technical units. |
| `lrSetAcc` | LREAL | Value of the acceleration (increasing energy of the motor) [u/s$^2$]. Only a positive value is allowed. |
| `lrSetDec` | LREAL | Value of the deceleration (decreasing energy of the motor) [u/s$^2$]. Only a positive value is allowed. |
| `eDirMovVelo` | SM3_Basic.MC_DIRECTION | Direction for continuous velocity operation:<br>● –1 = negative<br>● 1 = positive<br>● 2 = the active direction |
| `xStatEnbl` | BOOL | Indicates whether the drive is powered and the quick stop mechanism is disabled. |
| `xHomeActv` | BOOL | Indicates that the homing operation is active. |
| `xAbsActv` | BOOL | Indicates that the absolute positioning operation is active. |

| Variable | Data type | Description |
|---|---|---|
| xVeloActv | BOOL | Indicates that the continuous velocity operation is active. |
| xRelActv | BOOL | Indicates that the relative positioning operation is active. |
| xErr | BOOL | Indicates that an error state exists. |
| xActlPosVld | BOOL | Indicates whether the value of lrActlPos is valid. |
| lrActlPos | LREAL | Position in technical units. |
| xActlVeloVld | BOOL | Indicates whether the value of lrActlVelo is valid. |
| lrActlVelo | LREAL | Velocity u/s |
| xErrIdVld | BOOL | Indicates whether the value of dwErrId is valid. |
| dwErrId | DWORD | Vendor-specific value of the axis error. |
| xAxisStatVld | BOOL | Indicates whether the value of eAxisStat is valid. |
| eAxisStat | SM3_Basic.SMC_AXIS_STATE | State of the axis according to PLCopen state diagram. |
| diSercosStat | DINT | Indicates the state of the Sercos interface. This variable is assigned to the System variable ([Sercos interface name].State) which provides the state of the interface. |
| xComOk | BOOL | Indicates that the Sercos communication is OK. |
| diHmiSetHomePos | DINT | Position to set if homing is finished. |
| diHmiSetVeloMovVelo | DINT | Velocity setpoint for velocity mode in u/s. |
| diHmiSetVeloMovAbs | DINT | Velocity setpoint for absolute positioning u/s. |
| diHmiSetPosMovAbs | DINT | Target position for absolute positioning in technical units. |
| diHmiSetVeloMovRel | DINT | Velocity setpoint for relative positioning u/s. |
| diHmiSetDistMovRel | DINT | Distance for relative positioning in technical units. |
| diHmiActlPos | DINT | Position in technical units. |
| diHmiActlVelo | DINT | Velocity in u/s. |
| diHmiAcc | DINT | Value of the deceleration (increasing energy of the motor) [u/$s^2$]. Only a positive value is allowed. |
| diHmiDec | DINT | Value of the deceleration (decreasing energy of the motor) [u/$s^2$]. Only a positive value is allowed. |
| diDiagCode | DINT | Contains the diagnostic code of the device. |
| sDiagMsg | STRING(39) | Contains the diagnostic message of the device. |
| sDiagExtMsg | STRING(14) | Contains the extended diagnostic message of the device. |

## Program - Prg_<name device module>

### Program Contained in the Lexium_32S_Sercos Device Module

The program is divided into three actions and is created in programming languages CFC (Continuous Function Chart) and ST (Structured Text). All actions are called on each program execution if the variable `GVL_<name device module>.xEnable` is set to TRUE.

- Action - A01_DiagDevice
- Action - A02_Ctrl_Lexium
- Action - A03_HMIVarConversion

### Action - A01_DiagDevice

The program code in this action provides information about the state of the device on the Sercos motion bus. Based on the configured working mode and the present working state of the axis, in conjunction with the `DiagCode` of the device and the state of the Sercos interface, the communication state is assigned to the corresponding variable which has been declared in the `GVL_<name device module>`. The state of the Sercos interface is indicated by the variable `diSercosState` and must be assigned to the corresponding variable of the application after the Device Module has been added.

If the Sercos state is 4 (cyclic operation), and the `DiagCode` does not indicate a communication error while the detected working state of the Lexium 32S is `Real` or equal to the configured working mode, the variable for the general communication state (`xComOk`) indicates TRUE.

For the extended diagnostics of the Sercos device, the parameters `DiagMsg`, `DiagExtMsg` and the `DiagCode` are assigned to the associated variables in the GVL of this Device Module.

### Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power`, enable/disable the power stage of the drive
- with the function block `MC_Reset`, reset the drive on an error detected
- with the function block `SMC3_ReinitDrive`, reinitialize the axis
- with the function block `MC_Stop`, stop operation on the drive
- with the function block `MC_Home`, initiate the homing mode
- with the function block `MC_MoveVelocity`, operate the drive with continuous velocity
- with the function block `MC_MoveAbsolute`, operate the drive with absolute positioning
- with the function block `MC_MoveRelative`, operate the drive with relative positioning
- with the function block `MC_ReadActualVelocity`, read the velocity
- with the function block `MC_ReadActualPosition`, read the position
- with the function block `MC_ReadAxisError`, obtain the error state of the drive
- with the function block `MC_ReadParameter`, obtain the status of the drive

## Action - A03_HMIVarConversion

Not all HMI devices support the datatype LREAL, therefore variables of datatype DINT have been declared with the same meaning. In this action, the HMI variables of type DINT are converted and assigned to the process variables of type LREAL and vice versa.

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this device module requires a SoMachine controller with a Sercos III interface.

Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can:

- select the Sercos master which manages the device
- map variables to physical inputs and outputs of your configuration

Variable selected for I/O mapping (input):

| Variable | Data type | Default value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdy` | BOOL | – | Indicates the state of Motor Circuit Breaker. |

**NOTE:** When the Device Module has been added to your application, configure carefully the **LXM32S** drive (`<module name>`) and the **SoftMotion** drive (`SM_<module name>`). The drive settings have to be adapted according to your hardware and application.

**⚠ WARNING**

**UNINTENDED EQUIPMENT OPERATION**

Configure the **LXM32S** drive and the associated **SoftMotion** drive by editing the default parameters to those that conform to both your hardware and application needs.

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

Double-click the **LXM32S** drive (`<module name>`) in the **Devices Tree** to open the appropriate device editor. In the tab **Sercos Cyclic Data Exchange**, configure the Sercos address and the operation mode.

Double-click the associated **SM_Drive** (`SM_<module name>`) in the **Devices Tree** to open the appropriate device editor.

Edit the default parameter in the following tabs of the **SM_Drive** device editor in accordance with your hardware and application:

- **SoftMotion Drive: Basic**
  Allows you to make the settings for axis type, limits, and velocity ramp type.
- **SoftMotion Drive: Scaling/Mapping**
  Allows you to define the scaling between motor encoder increments and units in application.

# Section 2.18
## Lexium_IL•2K_EtherNetIP Device Modules

### Overview

This section provides a generic description for the following Device Modules:

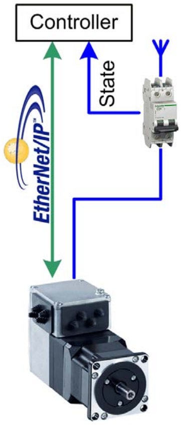
- Lexium_ILA2K_EtherNetIP
- Lexium_ILE2K_EtherNetIP
- Lexium_ILS2K_EtherNetIP

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_IL•2K_EtherNetIP Device Module Description

The Device Module Lexium_IL•2K_EtherNetIP provides the application objects and the device which are required to monitor and control an integrated Lexium IL• via EtherNet/IP with a Schneider Electric SoMachine controller. The device Lexium IL• requires the **Industrial Ethernet manager** under the Ethernet interface of the controller.

# Required Libraries

## Required Libraries Used in the Lexium_IL•2K_EtherNetIP Device Module

The Device Module implements objects from one or more libraries. The objects and the associated libraries used by this Device Module are listed in the following tables.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power | GMC Independent PLCopen MC | GIPLC | Schneider Electric |
| MC_Reset | | | |
| MC_Stop | | | |
| MC_MoveVelocity | | | |
| MC_MoveRelative | | | |
| MC_MoveAbsolute | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadActualPosition | | | |
| MC_ReadStatus | | | |
| MC_ReadAxisInfo | | | |
| MC_ReadAxisError | | | |
| Home_ILX | GMC Independent Lexium | GILXM | |
| Jog_ILX | | | |
| SetLimitSwitch_ILX | | | |
| SetDriveRamp_ILX | | | |
| SetStopRamp_ILX | | | |
| StoreParameters_ILX | | | |
| EIPGetHealthBit | EtherNetIP Scanner | EIPSC | |
| EIPStartConnection | | | |
| EIPStopConnection | | | |
| FB_RemoteAdapter | EtherNetIP Remote Adapter | EIPRA | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| `OperationErrorCodes` | EtherNetIP Scanner | EIPSC | Schneider Electric |
| `CIPOperationErrorCodes` | | | |
| `CommunicationErrorCodes` | | | |
| `eStatus` | EtherNetIP Remote Adapter | EIPRA | |
| `eAdapterErrorInfo` | | | |

**NOTE:** The library EtherNetIP Scanner is not supported by the motion controller LMC078.

# Functional Description

## Device - <name device module>

The Device Module implements the device Lexium IL• for EtherNet/IP. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** dialog box.

The device is preconfigured. The configuration includes the connection for the drive profile Lexium with the assemblies 103 (output) and 113 (input). The Request Packet Interval (RPI) is selected with 10 ms.

## Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

## Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

For basic control functions, the program code does not need to be modified, all required signals and parameter are linked to the associated variables in the GVL.

The program is divided into several actions. These are described in the following table.

**NOTE:** The program logic of the action `A01_ComCtrl` is not supported in an application of a motion controller LMC078.

| Name of the action | Description |
|---|---|
| `A01_ComCtrl` | Processes the functions to monitoring and control of the EtherNet/IP communication with the device. |
| `A02_Ctrl_LXM` | Contains a selection of function block calls to control the Lexium. Each function block is called in each program cycle. |
| `A03_Stat_LXM` | Contains a selection of function block calls to gather status information from the Lexium. Each function block is called in each program cycle. |
| `A04_Config_LXM` | Contains a selection of function block calls to write a set of parameters to the Lexium. |

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_EtherNetIP. For more information, refer to Adding Device Module to the Project *(see page 33)*.

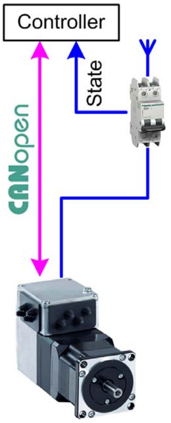
# Section 2.19
## Lexium_ILA_CANopen Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_ILA_CANopen Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium ILA via CANopen through a SoMachine controller.

The Device Module Lexium_ILA_CANopen is represented by a function template and consists of a global variable list, a program, and the device Lexium ILA under the CANopen manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the Lexium ILA via CANopen.

The program provides the following features:
- monitor the communication state of the device
- monitor the state of the device
- control the device in jog mode
- control the device in velocity mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

# Required Libraries

## Required Libraries Used in the Lexium_ILA_CANopen Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power_ILX | Integrated Lexium library | SEM_ILX | Schneider Electric |
| MC_Reset_ILX | | | |
| MC_Stop_ILX | | | |
| MC_Jog_ILX | | | |
| MC_MoveVelocity_ILX | | | |
| MC_MoveAbsolute_ILX | | | |
| MC_ReadActualVelocity_ILX | | | |
| MC_ReadActualPosition_ILX | | | |
| MC_ReadAxisError_ILX | | | |
| MC_Home_ILX | | | |
| GET_STATE | CAA CiA 405 | CIA405 | CAA Technical Workgroup |

# Global Variable List - GVL_<name device module>

## Gobal Variables Provided by the Lexium_ILA_CANopen Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xMcbRdy | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| iActlVelo | INT | Indicates the velocity of the drive. |
| diActlPos | DINT | Indicates the position of the drive. |
| xCmdEnPwr | BOOL | Enables the power stage of the drive. |
| xCmdRst | BOOL | Resets the drive in case of an error state. |
| xCmdStop | BOOL | Stops the drive. |
| xCmdJogFwd | BOOL | Jogs the drive in a forward direction. |
| xCmdJogRev | BOOL | Jogs the drive in a reverse direction. |
| xCmdJogFast | BOOL | Defines the velocity setpoint for jog operation. |
| uJogDist | UINT | Defines the distance to move for one interval on jog operation. If the value is set to 0, continuous motion is used. |
| uiWaitTimeJog | UINT | Defines the time delay in ms for change to continuous motion. |
| iSetVeloJogSlow | INT | Velocity setpoint for jog operation at slow speed. |
| iSetVeloJogFast | INT | Velocity setpoint for jog operation at fast speed. |
| xCmdMovVelo | BOOL | Starts the drive with continuous velocity |
| xCmdMovAbs | BOOL | Starts the drive for absolute positioning |
| iSetVeloMovVelo | INT | Velocity setpoint for velocity mode in rpm. |
| iSetVeloMovAbs | INT | Velocity setpoint for absolute positioning in rpm. |
| iVeloType | INT | Specification of the source of the velocity. |
| diSetPosMovAbs | DINT | Target position for absolute positioning in increments. |
| xCmdHoming | BOOL | Starts homing operation. |
| diSetHomePos | DINT | Position to set if homing is finished. |
| uiSetHomeMod | UINT | Defines the method for homing operation. |
| uiSetVeloHome | UINT | Velocity setpoint for search of the reference switch. |
| uiSetVeloOutHome | UINT | Velocity setpoint for movement back to edge of reference switch. |
| diSetPosOutHome | DINT | Maximum distance for movement back to edge of reference switch. |
| diSetPosDisHome | DINT | Distance for positioning starting from edge of reference switch. |
| iPosType | INT | Specification of the source of the position. |

| Variable | Data Type | Description |
|---|---|---|
| `xStatEnbl` | BOOL | Indicates the state of the power stage. |
| `wErrID` | WORD | Indicates the error ID of the detected error. Refer to the *Lexium Library Function Blocks Software Manual*. |
| `xErr` | BOOL | Indicates that an error state exists. |
| `xVeloActv` | BOOL | Indicates the continuous velocity operation is active. |
| `xAbsActv` | BOOL | Indicates the absolute positioning operation is active. |
| `xHomeActv` | BOOL | Indicates the homing operation is active. |
| `xJogActv` | BOOL | Indicates the jogging operation is active. |
| `xComOk` | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| `eComStat` | CIA405.DEVICE_STATE | Communication state of the device. Enumeration, refer to the CIA405 Library Guide (see SoMachine Online Help under *CoDeSys Libraries/CAA Libraries/CAA_CiA405.library*). |

## Program - Prg_<name device module>

### Program Contained in the Lexium_ILA_CANopen Device Module

The program is divided into 2 actions and is created in programming language CFC (Continuous Function Chart). Both actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_Lexium

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANopen fieldbus. Based on the communication parameter the communication state is assigned to the corresponding variables which have been declared in the GVL_<module name>. If the state is equal to `OPERATIONAL`, the variable for the state indicates TRUE and other cases are indicated by FALSE.

### Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power_ILX`, enable/disable the power stage of the drive
- with the function block `MC_Reset_ILX`, reset the drive after an error
- with the function block `MC_Stop_ILX`, stop operation on the drive
- with the function block `MC_Jog_ILX`, operate the drive in jog mode
- with the function block `MC_MoveVelocity_ILX`, operate the drive with continuous velocity
- with the function block `MC_MoveAbsolute_ILX`, operate the drive with absolute positioning
- with the function block `MC_ReadActualVelocity_ILX`, read the velocity of the drive
- with the function block `MC_ReadActualPosition_ILX`, read the position of the drive
- with the function block `MC_ReadAxisError_ILX`, obtain the error state of the drive
- with the function block `MC_Home_ILX`, initiate the homing mode

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_CANopen. For more information, refer to Adding Device Module to the Project *(see page 47)*.

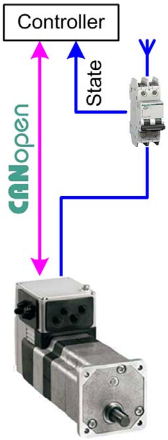
# Section 2.20
## Lexium_ILE_CANopen Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_ILE_CANopen Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium ILE via CANopen through a SoMachine controller.

The Device Module Lexium_ILE_CANopen is represented by a function template and consists of a global variable list, a program, and the device Lexium ILE under the CANopen manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the Lexium ILE via CANopen.

The program provides the following features:
- monitor the communication state of the device
- monitor the state of the device
- control the device in jog mode
- control the device in velocity mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

# Required Libraries

## Required Libraries Used in the Lexium_ILE_CANopen Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power_ILX | Integrated Lexium Library | SEM_ILX | Schneider Electric |
| MC_Reset_ILX | | | |
| MC_Stop_ILX | | | |
| MC_Jog_ILX | | | |
| MC_MoveVelocity_ILX | | | |
| MC_MoveAbsolute_ILX | | | |
| MC_ReadAxisError_ILX | | | |
| MC_ReadActualVelocity_ILX | | | |
| MC_ReadActualPosition_ILX | | | |
| MC_Home_ILX | | | |
| GET_STATE | CAA CiA 405 | CIA405 | CAA Technical Workgroup |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Lexium_ILE_CANopen Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xMcbRdy | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| iActlVelo | INT | Indicates the velocity of the drive. |
| diActlPos | DINT | Indicates the position of the drive. |
| xCmdEnPwr | BOOL | Enables the power stage of the drive. |
| xCmdRst | BOOL | Resets the drive in case of an error state. |
| xCmdStop | BOOL | Stops the drive. |
| xCmdJogFwd | BOOL | Jogs the drive in a forward direction. |
| xCmdJogRev | BOOL | Jogs the drive in a reverse direction. |
| xCmdJogFast | BOOL | Defines the velocity setpoint for jog operation. |
| uiSetJogDist | UINT | Defines the distance to move for one interval on jog operation. If the value is set to 0, continuous motion is used. |
| uiWaitTimeJog | UINT | Defines the time delay in ms for change to continuous motion. |
| iSetVeloJogSlow | INT | Velocity setpoint for jog operation at slow speed. |
| iSetVeloJogFast | INT | Velocity setpoint for jog operation velocity at fast speed. |
| xCmdMovVelo | BOOL | Starts the drive with continuous velocity |
| xCmdMovAbs | BOOL | Starts the drive for absolute positioning |
| iSetVeloMovVelo | INT | Velocity setpoint for velocity mode in rpm. |
| iSetVeloMovAbs | INT | Velocity setpoint for absolute positioning in rpm. |
| iVeloType | INT | Specification of the source of the velocity. |
| diSetPosMovAbs | DINT | Target position for absolute positioning in increments. |
| xCmdHoming | BOOL | Starts homing operation. |
| diSetHomePos | DINT | Position to set if homing is finished. |
| uiSetHomeMod | UINT | Defines the method for homing operation. |
| uiSetVeloHome | UINT | Velocity setpoint for search of the reference switch. |
| uiSetVeloOutHome | UINT | Velocity setpoint for movement back to edge of reference switch. |
| diSetPosOutHome | DINT | Maximum distance for movement back to edge of reference switch. |
| diSetPosDisHome | DINT | Distance for positioning starting from edge of reference switch. |
| iPosType | INT | Specification of the source of the position. |

| Variable | Data Type | Description |
|---|---|---|
| `xStatEnbl` | BOOL | Indicates the state of the power stage. |
| `wErrID` | WORD | Indicates the error ID of the detected error. Refer to the *Lexium Library Function Blocks Software Manual*. |
| `xErr` | BOOL | Indicates that an error state exists. |
| `xVeloActv` | BOOL | Indicates the continuous velocity operation is active. |
| `xAbsActv` | BOOL | Indicates the absolute positioning operation is active. |
| `xHomeActv` | BOOL | Indicates the homing operation is active. |
| `xJogActv` | BOOL | Indicates the jogging operation is active. |
| `xComOk` | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE =  communication state not operational |
| `eComStat` | CIA405.DEVICE_STATE | Communication state of the device. Enumeration, refer to the CIA405 Library Guide (see SoMachine Online Help under *CoDeSys Libraries/CAA Libraries/CAA_CiA405.library*). |

# Program - Prg_<name device module>

## Program Contained in the Lexium_ILE_CANopen Device Module

The program is divided into 2 actions and is created in programming language CFC (Continuous Function Chart). Both actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_Lexium

## Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANopen fieldbus. Based on the communication parameter the communication state is assigned to the corresponding variables which have been declared in the GVL_<module name>. If the state is equal to `OPERATIONAL`, the variable for the state indicates TRUE and other cases are indicated by FALSE.

## Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power_ILX`, enable/disable the power stage of the drive
- with the function block `MC_Reset_ILX`, reset the drive after an error
- with the function block `MC_Stop_ILX`, stop operation on the drive
- with the function block `MC_Jog_ILX`, operate the drive in jog mode
- with the function block `MC_MoveVelocity_ILX`, operate the drive with continuous velocity
- with the function block `MC_MoveAbsolute_ILX`, operate the drive with absolute positioning
- with the function block `MC_ReadActualVelocity_ILX`, read the velocity of the drive
- with the function block `MC_ReadActualPosition_ILX`, read the position of the drive
- with the function block `MC_ReadAxisError_ILX`, obtain the error state of the drive
- with the function block `MC_Home_ILX`, initiate the homing mode

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module ATV•••_CANopen. For more information, refer to Adding Device Module to the Project *(see page 47)*.

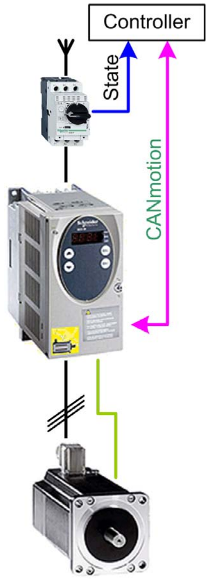
# Section 2.21
## Lexium_SD3_CANmotion Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Lexium_SD3_CANmotion Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a Lexium SD3 via CANmotion through a SoMachine controller.

The Device Module Lexium_SD3_CANmotion is represented by a function template and consists of a global variable list, a program, and the device Lexium SD3 under the CANmotion manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the Lexium SD3 via CANmotion.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device in velocity mode
- control the device in relative positioning mode
- control the device in absolute positioning mode
- control the device in homing mode
- reset the drive in case of an error state

# Required Libraries

## Required Libraries Used in the Lexium_SD3_CANmotion Device Module

The following function blocks are used in the program organization units (POUs) of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MC_Power | SM3_Basic | SM3_Basic | 3S - Smart Software Solutions GmbH |
| MC_Reset | | | |
| MC_Stop | | | |
| MC_Jog | | | |
| MC_MoveVelocity | | | |
| MC_MoveRelative | | | |
| MC_MoveAbsolute | | | |
| MC_ReadActualVelocity | | | |
| MC_ReadActualPosition | | | |
| MC_ReadAxisError | | | |
| MC_ReadStatus | | | |
| SMC3_ReinitDrive | | | |

## Global Variable List - GVL_<name device module>

### Global Variables Provided by the Lexium_SD3_CANmotion Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xMcbRdy | BOOL | Indicates the state of Motor Circuit Breaker (MCB). Mapped to the physical input where the corresponding auxiliary contact of the MCB is connected. |
| xCmdEnPwr | BOOL | Enables power to the drive. |
| xQuickStop | BOOL | Disables the quick stop mechanism. |
| xCmdRst | BOOL | Resets an error state on the axis. |
| xCmdReinitDrive | BOOL | Reinitializes the axis (start-up phase is reactivated) |
| xCmdStop | BOOL | Stops the axis |
| xCmdHoming | BOOL | Starts homing operation |
| xCmdMovVelo | BOOL | Starts the axis with continuous velocity |
| xCmdMovAbs | BOOL | Starts the axis with absolute positioning |
| xCmdMovRel | BOOL | Starts the axis with relative positioning |
| lrSetHomePos | LREAL | Position to set if homing is finished. |
| lrSetVeloMovVelo | LREAL | Velocity setpoint for velocity mode in u/s. |
| lrSetVeloMovAbs | LREAL | Velocity setpoint for absolute positioning u/s. |
| lrSetPosMovAbs | LREAL | Target position for absolute positioning in technical units. |
| lrSetVeloMovRel | LREAL | Velocity setpoint for relative positioning u/s. |
| lrSetDistMovRel | LREAL | Distance for relative positioning in technical units. |
| lrSetAcc | LREAL | Value of the acceleration [u/s$^2$] |
| lrSetDec | LREAL | Value of the deceleration [u/s$^2$] |
| eDirMovVelo | SM3_Basic.MC_DIRECTION | Direction for continuous velocity operation<br>-1 = negative<br>1 = positive<br>2 = the active direction |
| xStatEnbl | BOOL | Indicates whether the drive is powered and quick stop mechanism is disabled |
| xHomeActv | BOOL | Indicates the homing operation is active. |
| xAbsActv | BOOL | Indicates the absolute positioning operation is active. |
| xVeloActv | BOOL | Indicates the continuous velocity operation is active. |
| xRelActv | BOOL | Indicates the relative positioning operation is active. |

| Variable | Data Type | Description |
|---|---|---|
| xErr | BOOL | Indicates that an error state exists. |
| xActlPosVld | BOOL | Indicates whether the value lrActlPos is valid. |
| lrActlPos | LREAL | Position of axis unit [u] |
| xActlVeloVld | BOOL | Indicates whether the value lrActlVelo is valid. |
| lrActlVelo | LREAL | Velocity of axis unit [u/s] |
| xErrIdVld | BOOL | Indicates whether the value dwErrId is valid. |
| dwErrId | DWORD | Vendor-specific value of the axis error |
| xAxisStatVld | BOOL | Indicates whether the value eAxisStat is valid. |
| eAxisStat | SM3_Basic.SMC_AXIS_STATE | State of the axis according to PLCopen state diagram |
| xComOk | BOOL | Indicates the CANmotion communication state.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| diHmiSetHomePos | DINT | Position to set if homing is finished. |
| diHmiSetVeloMovVelo | DINT | Velocity setpoint for velocity mode in u/s. |
| diHmiSetVeloMovAbs | DINT | Velocity setpoint for absolute positioning u/s. |
| diHmiSetPosMovAbs | DINT | Target position for absolute positioning in technical units. |
| diHmiSetVeloMovRel | DINT | Velocity setpoint for relative positioning u/s. |
| diHmiSetDistMovRel | DINT | Distance for relative positioning in technical. units. |
| diHmiActlPos | DINT | Position of axis unit [u] |
| diHmiActlVelo | DINT | Velocity of axis unit [u/s] |
| diHmiAcc | DINT | Value of the acceleration [u/s$^2$] |
| diHmiDec | DINT | Value of the deceleration [u/s$^2$] |

## Program - Prg_<name device module>

### Program Contained in the Lexium_SD3_CANmotion Device Module

The program is divided into three actions and is created in programming language CFC (Continuous Function Chart). These actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_Lexium
- Action - A03_HmiVarConversion

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANmotion bus. The communication state is provided by an element of the axis structure and is assigned to the corresponding variable which has been declared in the `GVL_<module name>.` If the CANmotion communication is OK, the variable for the general state indicates TRUE, otherwise the state of the variable is FALSE.

### Action - A02_Ctrl_Lexium

The program code in this action processes the basic monitor and control functions of the device:

- with the function block `MC_Power`, enable/disable the power stage of the drive
- with the function block `MC_Reset`, reset the drive after an error
- with the function block `SMC3_ReinitDrive`, reinitialize the axis
- with the function block `MC_Stop`, stop operation on the drive
- with the function block `MC_Home`, initiate the homing mode
- with the function block `MC_MoveVelocity`, operate the drive with continuous velocity
- with the function block `MC_MoveAbsolute`, operate the drive with absolute positioning
- with the function block `MC_MoveRelative`, operate the drive with relative positioning
- with the function block `MC_ReadActualVelocity`, read the velocity
- with the function block `MC_ReadActualPosition`, read the position
- with the function block `MC_ReadAxisError`, obtain the error state of the drive
- with the function block `MC_ReadParameter`, obtain the status of the drive

### Action - A03_HmiVarConversion

Not all HMI devices support the datatype LREAL, therefore variables of datatype DINT have been declared with the same meaning. In this action, the HMI variables of type DINT will be converted and assigned to the process variables of type LREAL.

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module is equivalent to the Device Module Lexium_32A_CAN-motion. For more information, refer to Adding Device Module to the Project *(see page 102)*.

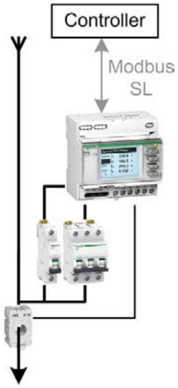
# Section 2.22
## MED_iEM3150_ModbusSL Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## MED_iEM3150_ModbusSL Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor energy data, such as current, voltage and power. The SoMachine controller retrieves the energy data from the Device Module via Modbus SL.

The Device Module MED_iEM3150_ModbusSL is represented by a function template and consists of a global variable list (GVL), and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which contain the energy data and additional information. These variables can be directly connected to the associated Machine Energy Dashboard widgets on the Magelis HMI. These widgets are provided within the Toolchest of Vijeo Designer.

After instantiation, a variable `wModbusToken` is added to the global variable list with the name `GVL`. This variable is used by the `FB_PowerMeter`. The FB checks this variable for value 0 to start the communication. During active communication the used slave address is written to the variable. When the communication is finished, the FB writes 0 to the variable. This variable is used to interlock other Modbus SL communication function blocks in the application.

The program provides the following features:

- read the energy data from the Energy Meter
- monitor the current
- monitor the voltage
- monitor the power
- monitor the power factor
- monitor the frequency
- monitor the total energy consumption
- monitor the energy consumption per mode
- monitor the Modbus communication

# Required Libraries

## Required Libraries Used in the MED_iEM3150_ModbusSL Device Module

The following function blocks and structures are used in the POUs of the template. The corresponding libraries are added to the project if the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| FB_PowerMeter | SE_ModbusEnergyEfficiencyToolbox | MEET | Schneider Electric |
| FB_EE_EnergyQuality | SE_MachineEnergyDashboard | MED | |
| FB_EE_PowerAndEnergy | | | |
| FB_EE_Frequency | | | |
| FB_EE_EnergyPerMode | | | |
| FB_EE_InstantPower | | | |
| FB_EE_TotalEnergy | | | |

| Structure | Library | Namespace | Vendor |
|---|---|---|---|
| ST_MdbCommParaGeneric | SE_ModbusEnergyEfficiencyToolbox | MEET | Schneider Electric |
| EqPhasVal | SE_MachineEnergyDashboard | MED | |
| EqWdgtConf | | | |
| FrVal | | | |
| EEEnrgy | | | |
| PAEVal | | | |
| PAEWdgtConf | | | |
| EpmWdgtConf | | | |
| IpPower | | | |
| IpWdgtConf | | | |
| IpScalPara | | | |
| ST_GenericDeviceDatasetLong | SE_EnergyEfficiencyToolbox | EET | |

## Global Variable List - GVL_<name device module>

### Global Variables Provided by the MED_iEM3150_ModbusSL Device Module

The table presents the variables provided with the global variable list (GVL):

| Variable | Data Type | Description |
|---|---|---|
| xCmdRestart | BOOL | Restarts FB which reads data from power meter, for example, during a communication interruption. |
| xCmdRst | BOOL | Reset FB which reads data from power meter. |
| xCmdRstFbMedCurr | BOOL | Reset the FB which processes the current values. |
| xCmdRstFbMedVltPP | BOOL | Reset the FB which processes the voltage (phase to phase) values. |
| xCmdRstFbMedVltgPN | BOOL | Reset the FB which processes the voltage (phase to neutral) values. |
| xCmdRstFbMedPwrTotal | BOOL | Reset the FB which processes the total power values. |
| xCmdRstFbMedPwrFact | BOOL | Reset the FB which processes the power factor values. |
| xCmdRstFbMedFreq | BOOL | Reset the FB which processes the frequency values. |
| xCmdRstFbMedTotalEnergy | BOOL | Reset the FB which processes the total energy values. |
| xCmdRstFbMedEnergyPerMode | BOOL | Reset the function blocks which process the energy per mode values. |
| xCmdRstFbMedInstantPwr | BOOL | Reset the FB which processes the instant power values. |
| xWdgtModeInstantPwr | BOOL | Switch the display between operator mode and maintenance mode. |
| xWdgtModeEnergyPerMode | BOOL | Switch the display between operator mode and maintenance mode. |
| xMachRun | BOOL | Indicates, the machine is in RUN mode for the monitoring of energy per mode. |
| stHmiWdgtValCurr | MED.EqWdgtVal | Current values to be displayed on the widget on the HMI. |
| stHmiWdgtParaCurr | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| xAlarmCurr | BOOL | Indicates an alarm, occurred in the associated FB. |
| stHmiWdgtValVltgPP | MED.EqWdgtVal | Voltage values (phase to phase) to be displayed on the widget on the HMI. |
| stHmiWdgtParaVltgPP | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| xAlarmVltgPP | BOOL | Indicates an alarm, occurred in the associated FB. |
| stHmiWdgtValVltgPN | MED.EqWdgtVal | Voltage values (phase to neutral) to be displayed on the widget on the HMI. |
| stHmiWdgtParaVltgPN | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |

| Variable | Data Type | Description |
|---|---|---|
| xAlarmVltgPN | BOOL | Indicates an alarm, occurred in the associated FB. |
| stHmiWdgtValPwrTotal | MED.PAEWdgtVal | Power values (total) to be displayed on the widget on the HMI. |
| stHmiWdgtParaPwrTotal | MED.PAEWdgtPara | Parameters determining the appearance of the associated widget. |
| stHmiWdgtValFreq | MED.FrWdgtVal | Frequency value to be displayed on the widget on the HMI. |
| xAlarmFreq | BOOL | Indicates that an alarm occurred in the associated FB. |
| stHmiWdgtValPwrFact | MED.EqWdgtVal | Power factor to be displayed on the widget on the HMI. |
| stHmiWdgtParaPwrFact | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| xAlarmPwrFact | BOOL | Indicates an alarm, occurred in the associated FB. |
| stHmiWdgtValTotalEnergy | MED.TEWdgtVal | Energy value consumed since the last reset of the FB to be displayed on the widget on the HMI. |
| stHmiWdgtConfTotalEnergy | MED.TEWdgtConf | Parameters determining the appearance of the associated widget. |
| xAlarmTotalEnergy | BOOL | Indicates that an alarm occurred in the associated FB. |
| stHmiWdgtValRunEnergy | MED.EpmWdgtVal | Energy value consumed in Run mode to be displayed on the widget on the HMI. |
| stHmiWdgtParaRunEnergy | MED.EpmWdgtPara | Parameters determining the appearance of the associated widget. |
| xAlarmRunEnergy | BOOL | Indicates an alarm, occurred in the associated FB. |
| stHmiWdgtValIdleEnergy | MED.EpmWdgtVal | Energy value consumed in Idle mode to be displayed on the widget on the HMI. |
| stHmiWdgtParaIdleEnergy | MED.EpmWdgtPara | Parameters determining the appearance of the associated widget. |
| xAlarmIdleEnergy | BOOL | Indicates an alarm, occurred in the associated FB. |
| stHmiWdgtValInstantPwr | MED.IpWdgtVal | Instant power value to be displayed on the widget on the HMI. |
| stHmiWdgtParaInstantPwr | MED.IPWdgtPara | Parameters determining the appearance of the associated widget. |
| xAlarmInstantPwr | BOOL | Indicates that an alarm occurred in the associated FB. |
| xComErr | BOOL | Indicates a Modbus communication interruption. Restart command for FB Power Meter is required. |

## Program - Prg_<name device module>

### Program Contained in the MED_iEM3150_ModbusSL Device Module

The program is created in programming language CFC (Continuous Function Chart) and includes 2 steps. One is the reading of energy data from the Energy Meter and the other one is the processing of the energy data for displaying on widgets on the HMI.

The program code for reading the energy data is executed in the program with the use of the FB `FB_PowerMeter`. The FB uses system functions internally to get the data via Modbus SL communication from the device. The communication state is monitored and indicated by an output of the FB.

The program code to process the energy values is divided into 8 actions. These actions will be called on each program execution.

### Action - A01_MED_Current

In this action, the current values are processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the current values and the configuration parameter for the widget on the HMI.

### Action - A02_MED_Voltage

In this action, the voltage values are processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the voltage values and the configuration parameter for the widget on the HMI.

### Action - A03_MED_Power

In this action, the power values are processed with the use of the FB `FB_EE_PowerAndEnergy`. This FB provides the power values and the configuration parameter for the widget on the HMI.

### Action - A04_MED_PowerFactor

In this action, the power factor is processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the power factor and the configuration parameter for the widget on the HMI.

### Action - A05_MED_Frequency

In this action, the frequency value is processed with the use of the FB `FB_EE_Frequency`. This FB provides the frequency value and the configuration parameter for the widget on the HMI.

### Action - A06_MED_Energy

In this action, the energy values are processed with the use of the function blocks `FB_EE_TotalEnergy` and `FB_EE_PowerAndEnergy`. Each FB provides the energy values and the configuration parameter for the widget on the HMI. The FB `FB_EE_PowerAndEnergy` provides the energy evaluated by the Energy Meter and the FB `FB_EE_TotalEnergy` provides the consumed energy since the last reset on the FB.

## Action - A07_MED_EnergyPerMode

In this action, the energy values of the respective machine modes are processed with the use of the FB `FB_EE_EnergyPerMode`. This FB provides the energy value and the configuration parameter for the widget on the HMI. 2 machine modes are defined within this Device Module: the run mode and the idle mode. The variable `GVL_<name device module>.xMachRun` determines the mode as follows.

- TRUE = Run mode
- FALSE = Idle mode

## Action - A08_MED_InstantPower

In this action, the power value is processed with the use of the FB `FB_EE_InstantPower`. This FB provides the power value, the monitoring state, and the configuration parameter for the widget on the HMI.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires a Modbus manager be added to the serial line interface of your controller. Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can adjust the initial values for selected variables which are part of the template.

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<module name>.c_byAddr` | BYTE | 1 | Modbus slave address of the Energy Meter |
| `Prg_<module name>.c_byChanNb` | BYTE | 1 | Communication port of the controller |
| `Prg_<module name>.c_iMinPower` | INT | 0 | Absolute value in Watt that defines the minimum value in the meter graph of the instant power object. |
| `Prg_<module name>.c_iMaxPower` | INT | 2000 | Absolute value in Watt that defines the maximum value in the meter graph of the instant power object. |

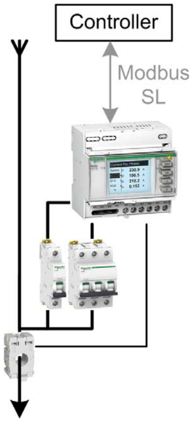
# Section 2.23
# MED_PM3250_ModbusSL Device Module

## What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## MED_PM3250_ModbusSL Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor energy data, such as current, voltage, and power. The SoMachine controller retrieves the energy data from the Device Module via Modbus SL.

The Device Module MED_PM3250_ModbusSL is represented by a function template and consists of a global variable list GVL, and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which contain the energy data and additional information. These variables can be directly connected to the associated Machine Energy Dashboard widgets on the Magelis HMI. These widgets are provided within the Toolchest of Vijeo Designer.

After instantiation, a variable `wModbusToken` is added to the global variable list with the name `GVL`. This variable is used by the `FB_PowerMeter`. The FB checks this variable for value 0 to start the communication. During active communication the used slave address is written to the variable. When the communication is finished, the FB writes 0 to the variable. This variable is used to interlock other Modbus SL communication function blocks in the application.

The program provides the following features:

- read the energy data from the Power Meter
- monitor the current
- monitor the voltage
- monitor the power
- monitor the power factor
- monitor the frequency
- monitor the total energy consumption
- monitor the energy consumption per mode
- monitor the harmonic content (THDI and THDU values)
- monitor the Modbus communication

# Required Libraries

## Required Libraries Used in the MED_PM3250_ModbusSL Device Module

The following function blocks and structures are used in the POUs of the template. The corresponding libraries are added to the project if the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| FB_PowerMeter | SE_ModbusEnergyEfficiencyToolbox | MEET | Schneider Electric |
| FB_EE_EnergyQuality | SE_MachineEnergyDashboard | MED | |
| FB_EE_PowerAndEnergy | | | |
| FB_EE_Frequency | | | |
| FB_EE_EnergyPerMode | | | |
| FB_EE_InstantPower | | | |
| FB_EE_TotalEnergy | | | |

| Structure | Library | Namespace | Vendor |
|---|---|---|---|
| ST_MdbCommParaGeneric | SE_ModbusEnergyEfficiencyToolbox | MEET | Schneider Electric |
| EqPhasVal | SE_MachineEnergyDashboard | MED | |
| EqWdgtConf | | | |
| FrVal | | | |
| EEEnrgy | | | |
| PAEVal | | | |
| PAEWdgtConf | | | |
| EpmWdgtConf | | | |
| IpPower | | | |
| IpWdgtConf | | | |
| IpScalPara | | | |
| ST_GenericDeviceDatasetLong | SE_EnergyEfficiencyToolbox | EET | |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the MED_PM3250_ModbusSL Device Module

The table presents the variables provided with the global variable list (GVL):

| Variable | Data Type | Description |
|---|---|---|
| xCmdRestart | BOOL | Restarts FB which reads data from power meter for example during a blocked communication. |
| xCmdRst | BOOL | Reset FB which reads data from power meter. |
| xCmdRstFbMedCurr | BOOL | Reset the FB which processes the current values. |
| xCmdRstFbMedVltPP | BOOL | Reset the FB which processes the voltage (phase to phase) values. |
| xCmdRstFbMedVltgPN | BOOL | Reset the FB which processes the voltage (phase to neutral) values. |
| xCmdRstFbMedPwrTotal | BOOL | Reset the FB which processes the total power values. |
| xCmdRstFbMedPwrL1 | BOOL | Reset the FB which processes reset the FB which processes the L1 power values. |
| xCmdRstFbMedPwrL2 | BOOL | Reset the FB which processes the L2 power values. |
| xCmdRstFbMedPwrL3 | BOOL | Reset the FB which processes the L3 power values. |
| xCmdRstFbMedPwrFact | BOOL | Reset the FB which processes the power factor values. |
| xCmdRstFbMedFreq | BOOL | Reset the FB which processes the frequency values. |
| xCmdRstFbMedEnergy | BOOL | Reset the FB which processes the detailed energy values. |
| xCmdRstFbMedTotalEnergy | BOOL | Reset the FB which processes the total energy values. |
| xCmdRstFbMedEnergyPerMode | BOOL | Reset the function blocks which processes the energy per mode values. |
| xCmdRstFbMedThdi | USINT | Reset the FB which processes the THDI values. |
| xCmdRstFbMedThdu | USINT | Reset the FB which processes the THDU values. |
| xCmdRstFbMedInstantPwr | BOOL | Reset the FB which processes the instant power values. |
| xWdgtModeInstantPwr | BOOL | Switch the display between operator mode and maintenance mode. |
| xWdgtModeEnergyPerMode | BOOL | Switch the display between operator mode and maintenance mode. |
| xMachRun | BOOL | Indicates, the machine is in RUN mode for the monitoring of energy per mode. |
| stHmiWdgtValCurr | MED.EqWdgtVal | Current values to be displayed on the widget on the HMI. |
| stHmiWdgtParaCurr | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| xAlarmCurr | BOOL | Indicates an alarm, occurred in the associated FB. |

| Variable | Data Type | Description |
|---|---|---|
| `stHmiWdgtValVltgPP` | MED.EqWdgtVal | Voltage values (phase to phase) to be displayed on the widget on the HMI. |
| `stHmiWdgtParaVltgPP` | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmVltgPP` | BOOL | Indicates an alarm, occurred in the associated FB. |
| `stHmiWdgtValVltgPN` | MED.EqWdgtVal | Voltage values (phase to neutral) to be displayed on the widget on the HMI. |
| `stHmiWdgtParaVltgPN` | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmVltgPN` | BOOL | Indicates an alarm, occurred in the associated FB. |
| `stHmiWdgtValPwrTotal` | MED.PAEWdgtVal | Power values (total) to be displayed on the widget on the HMI. |
| `stHmiWdgtParaPwrTotal` | MED.PAEWdgtPara | Parameters determining the appearance of the associated widget. |
| `stHmiWdgtValPwrL1` | MED.PAEWdgtVal | Power values (L1) to be displayed on the widget on the HMI. |
| `stHmiWdgtParaPwrL1` | MED.PAEWdgtPara | Parameters determining the appearance of the associated widget. |
| `stHmiWdgtValPwrL2` | MED.PAEWdgtVal | Power values (L2) to be displayed on the widget on the HMI. |
| `stHmiWdgtParaPwrL2` | MED.PAEWdgtPara | Parameters determining the appearance of the associated widget. |
| `stHmiWdgtValPwrL3` | MED.PAEWdgtVal | Power values (L3) to be displayed on the widget on the HMI. |
| `stHmiWdgtParaPwrL3` | MED.PAEWdgtPara | Parameters determining the appearance of the associated widget. |
| `stHmiWdgtValFreq` | MED.FrWdgtVal | Frequency value to be displayed on the widget on the HMI. |
| `xAlarmFreq` | BOOL | Indicates that an alarm occurred in the associated FB. |
| `stHmiWdgtValPwrFact` | MED.EqWdgtVal | Power factor to be displayed on the widget on the HMI. |
| `stHmiWdgtParaPwrFact` | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmPwrFact` | BOOL | Indicates an alarm, occurred in the associated FB. |
| `stHmiWdgtValEnergy` | MED.PAEWdgtVal | Energy values to be displayed on the widget on the HMI. |
| `stHmiWdgtParaEnerg` | MED.PAEWdgtPara | Parameters determining the appearance of the associated widget. |
| `stHmiWdgtValTotalEnergy` | MED.TEWdgtVal | Energy value consumed since the last reset of the FB to be displayed on the widget on the HMI. |
| `stHmiWdgtConfTotalEnergy` | MED.TEWdgtConf | Parameters determining the appearance of the associated widget. |

| Variable | Data Type | Description |
|---|---|---|
| `xAlarmTotalEnergy` | BOOL | Indicates that an alarm occurred in the associated FB. |
| `stHmiWdgtValRunEnergy` | MED.EpmWdgtVal | Energy value consumed in Run mode to be displayed on the widget on the HMI. |
| `stHmiWdgtParaRunEnergy` | MED.EpmWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmRunEnergy` | BOOL | Indicates an alarm, occurred in the associated FB. |
| `stHmiWdgtValIdleEnergy` | MED.EpmWdgtVal | Energy value consumed in Idle mode to be displayed on the widget on the HMI. |
| `stHmiWdgtParaIdleEnergy` | MED.EpmWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmIdleEnergy` | BOOL | Indicates an alarm, occurred in the associated FB. |
| `stHmiWdgtValThdu` | MED.EqWdgtVal | THDU value to be displayed on the widget on the HMI. |
| `stHmiWdgtParaThdu` | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmThdu` | BOOL | Indicates that an alarm occurred in the associated FB. |
| `stHmiWdgtValThdi` | MED.EqWdgtVal | THDI value to be displayed on the widget on the HMI. |
| `stHmiWdgtParaThdi` | MED.EqWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmThdi` | BOOL | Indicates an alarm, occurred in the associated FB. |
| `stHmiWdgtValInstantPwr` | MED.IpWdgtVal | Instant power value to be displayed on the widget on the HMI. |
| `stHmiWdgtParaInstantPwr` | MED.IPWdgtPara | Parameters determining the appearance of the associated widget. |
| `xAlarmInstantPwr` | BOOL | Indicates that an alarm occurred in the associated FB. |
| `xComErr` | BOOL | Indicates that the Modbus connection is blocked. Restart command for FB Power Meter is required. |

## Program - Prg_<name device module>

### Program Contained in the MED_PM3250_ModbusSL Device Module

The program is created in programming language CFC (Continuous Function Chart) and includes 2 steps. One is the reading of energy data from the Power Meter and the other one is the processing of the energy data for displaying on widgets on the HMI.

The program code for reading the energy data is executed in the program with the use of the FB `FB_PowerMeter`. The FB uses system functions internally to get the data via Modbus SL communication from the device. The communication state is monitored and indicated by an output of the FB.

The program code to process the energy values is divided into 9 actions. These actions will be called on each program execution.

### Action - A01_MED_Current

In this action, the current values are processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the current values and the configuration parameter for the widget on the HMI.

### Action - A02_MED_Voltage

In this action, the voltage values are processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the voltage values and the configuration parameter for the widget on the HMI.

### Action - A03_MED_Power

In this action, the power values are processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the power values and the configuration parameter for the widget on the HMI.

### Action - A04_MED_PowerFactor

In this action, the power factor is processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the power factor and the configuration parameter for the widget on the HMI.

### Action - A05_MED_Frequency

In this action, the frequency value is processed with the use of the FB `FB_EE_Frequency`. This FB provides the frequency value and the configuration parameter for the widget on the HMI.

### Action - A06_MED_Energy

In this action, the energy values are processed with the use of the function blocks `FB_EE_TotalEnergy` and `FB_EE_PowerAndEnergy`. Each FB provides the energy values and the configuration parameter for the widget on the HMI. The FB `FB_EE_PowerAndEnergy` provides the energy evaluated by the Power Meter and the FB `FB_EE_TotalEnergy` provides the consumed energy since the last reset on the FB.

### Action - A07_MED_EnergyPerMode

In this action, the energy values of the respective machine modes are processed with the use of the FB `FB_EE_EnergyPerMode`. This FB provides the energy value and the configuration parameter for the widget on the HMI. 2 machine modes are defined within this Device Module: the run mode and the idle mode. The variable `GVL_<name device module>.xMachRun` determines the mode as follows.

- TRUE = Run mode
- FALSE = Idle mode

### Action - A08_MED_TotalHarmonicDistortion

In this action, the total harmonic distortion values are processed with the use of the FB `FB_EE_EnergyQuality`. This FB provides the THDU respectively the THDI value and the configuration parameter for the widget on the HMI.

### Action - A09_MED_InstantPower

In this action, the power value is processed with the use of the FB `FB_EE_InstantPower`. This FB provides the power value, the monitoring state, and the configuration parameter for the widget on the HMI.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires that a Modbus manager be added to the serial line interface of your controller. Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can adjust the initial values for selected variables which are part of the template.

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<module name>.c_byAddr` | BYTE | 1 | Modbus slave address of the Power Meter |
| `Prg_<module name>.c_byChanNb` | BYTE | 1 | Communication port of the controller |
| `Prg_<module name>.c_iMinPower` | INT | 0 | Absolute value in Watt that defines the minimum value in the meter graph of the instant power object. |
| `Prg_<module name>.c_iMaxPower` | INT | 2000 | Absolute value in Watt that defines the maximum value in the meter graph of the instant power object. |

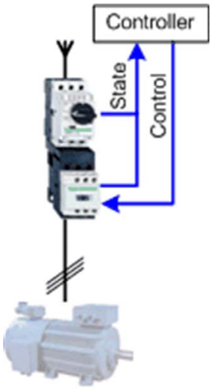
# Section 2.24
## Motor_Ctrl_1D1S Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Motor_Ctrl_1D1S Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a hardwired direct online motor starter through a SoMachine controller.

The Device Module Motor_Ctrl_1D1S is represented by a function template and consists of a global variable list (GVL), and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control a motor via hardwired I/Os in one direction with one speed.

The program provides the following features:
- monitor the state of the motor starter
- control the motor in manual mode (latch mode)
- control the motor in local mode (latch mode)
- control the motor in auto mode (jog mode)

# Required Libraries

## Required Libraries Used in the Motor_Ctrl_1D1S Device Module

The following function block is used in the program organization units (POU) of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| Mot2D1S | TeSys Library | SE_TESYS | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Motor_Ctrl_1D1S Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xSelAutMode | BOOL | Selects auto mode for the FB. |
| xSelManMode | BOOL | Selects manual mode for the FB. |
| xCmdAutFwd | BOOL | Starts (jog mode) the motor during auto mode. |
| xCmdLocFwd | BOOL | Local start (latch mode) of the motor in a forward direction during manual mode. |
| xCmdLocStop | BOOL | Local stop of the motor during manual mode. |
| xCmdManFwd | BOOL | Starts (latch mode) the motor in a forward direction during manual mode. |
| xCmdManStop | BOOL | Stops the motor during manual mode. |
| xCmdErrRst | BOOL | Resets the FB in case of alarm state. |
| xExtLock | BOOL | External signal to lock the FB (for example, state of the emergency stop). |
| xExtErr | BOOL | External signal to set the FB into error state (reset required). |
| xStatAutMode | BOOL | FB is selected for auto mode. |
| xStatManMode | BOOL | FB is selected for manual mode. |
| xStatLocMode | BOOL | FB is selected for manual and local mode. |
| xStatErr | BOOL | FB is in error state, reset required. |
| xAlertLock | BOOL | FB is blocked by i_xLock. |
| xAlarmOpMode | BOOL | Invalid operation mode selection has been done. |
| xAlarmExt | BOOL | FB is in alarm state due to detected i_xErr. |
| xAlarmFbckTmout | BOOL | FB is in alarm state due to undetected i_xFbckRun during the monitoring time (only if feedback signal supervision is activated). |
| xMcbRdy | BOOL | Signal associated with the motor circuit breaker contact indicating that the device is under power. |
| xFwdFbck | BOOL | Feedback signal indicating motor runs in a forward direction. |
| xDriveMotFwd | BOOL | Activates the contactor in a forward direction. |

## Program - Prg_<name device module>

### Program Contained in the Motor_Ctrl_1D1S Device Module

The program is created in programming language CFC (Continuous Function Chart).

Implemented features are:

- Mapping the manual commands into the control word.
- FB instance (`MOT2D1S`) call with assigned parameters.
- Extracting of the status word (detailed alarm and alert information) to boolean variables.

## Adding Device Module to the Project

### Instantiation of the Device Module

Using **Add Function From Template** *(see SoMachine, Programming Guide)* for this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdy` | BOOL | – | Signal associated with the motor circuit breaker contact indicating that the device is under power. |
| `GVL_<modul name>.xFwdFbck` | BOOL | – | Feedback signal indicating motor runs in a forward direction. |

Variable selected for I/O mapping (output):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xDriveMotFwd` | BOOL | – | Activates the drive in a forward direction. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.c_xEnFbckCtrl` | BOOL | FALSE | Enables the monitoring of the feedback signals of the motor run state. |
| `Prg_<modul name>.c_ iDlyTimeFbckCtrl` | INT | 2 | Delay time in seconds to determine that the feedback signal is inoperable and to activate an alarm. |

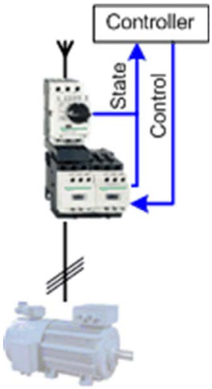
# Section 2.25
## Motor_Ctrl_2D1S Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Motor_Ctrl_2D1S Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a hardwired direct online motor starter in two directions through a SoMachine controller.

The Device Module Motor_Ctrl_2D1S is represented by a function template and consists of a global variable list (GVL), and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control a motor via hardwired I/Os in 2 directions with one speed.

The program provides the following features:
- monitor the state of the motor starter
- control the motor in manual mode (latch mode)
- control the motor in local mode (latch mode)
- control the motor in auto mode (jog mode)

# Required Libraries

## Required Libraries Used in the Motor_Ctrl_2D1S Device Module

The following function block is used in the program organization units (POU) of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| MOT2D1S | TeSys library | SE_TESYS | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the Motor_Ctrl_2D1S Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xSelAutMode | BOOL | Selects auto mode for the FB. |
| xSelManMode | BOOL | Selects manual mode for the FB. |
| xCmdAutFwd | BOOL | Starts (jog mode) the motor in a forward direction during auto mode. |
| xCmdAutRev | BOOL | Starts (jog mode) the motor in a reverse direction during auto mode. |
| xCmdLocFwd | BOOL | Local start (latch mode) of the motor in a forward direction during manual mode. |
| xCmdLocRev | BOOL | Local start (latch mode) of the motor in a reverse direction during manual mode. |
| xCmdLocStop | BOOL | Local stop of the motor during manual mode. |
| xCmdManFwd | BOOL | Starts (latch mode) the motor in a forward direction during manual mode. |
| xCmdManRev | BOOL | Starts (latch mode) the motor in a reverse direction during manual mode. |
| xCmdManStop | BOOL | Stops the motor during manual mode. |
| xCmdErrRst | BOOL | Resets the FB in case of alarm state. |
| xExtLock | BOOL | External signal to lock the FB (for example state of the emergency stop). |
| xExtErr | BOOL | External signal to set the FB into error state (reset required). |
| xStatAutMode | BOOL | FB is selected for auto mode. |
| xStatManMode | BOOL | FB is selected for manual mode. |
| xStatLocMode | BOOL | FB is selected for manual & local mode. |
| xStatErr | BOOL | FB is in error state, reset required. |
| xAlertLock | BOOL | FB is blocked by i_xLock. |
| xAlarmOpMode | BOOL | Invalid operation mode selection has been done. |
| xAlarmExt | BOOL | FB is in alarm state due to detected i_xErr. |
| xAlarmFbckTmout | BOOL | FB is in alarm state due to undetected i_xFwdFbck or i_xRevFbck during the monitoring time (only if feedback signal supervision is activated). |
| xMcbRdy | BOOL | Signal associated with the motor circuit breaker contact indicating that the device is under power. |
| xFwdFbck | BOOL | Feedback signal indicating motor runs in a forward direction. |
| xRevFbck | BOOL | Feedback signal indicating motor runs in a reverse direction. |
| xDriveMotFwd | BOOL | Activates the contactor for the forward direction. |
| xDriveMotRev | BOOL | Activates the contactor for the reverse direction. |

## Program - Prg_<name device module>

### Program Contained in the Motor_Ctrl_2D1S Device Module

The program is created in programming language CFC (Continuous Function Chart).

Implemented features are:

1. Mapping the manual command to the control word.
2. FB instance (`MOT2D1S`) call with assigned parameters.
3. Extracting of the status word (detailed alarm and alert information) to boolean variables.

# Adding Device Module to the Project

## Instantiation of the Device Module

Using **Add Function from Template** *(see SoMachine, Programming Guide)* for this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdy` | BOOL | – | Signal associated with the motor circuit breaker contact indicating that the device is under power. |
| `GVL_<modul name>.xFwdFbck` | BOOL | – | Feedback signal indicating motor runs in a forward direction. |
| `GVL_<modul name>.xRevFbck` | BOOL | – | Feedback signal indicating motor runs in a reverse direction. |

Variables selected for I/O mapping (output):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xDriveMotFwd` | BOOL | – | Activates the contactor for the forward direction. |
| `GVL_<modul name>.xDriveMotRev` | BOOL | – | Activates the contactor for the reverse direction. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.c_xEnFbckCtrl` | BOOL | FALSE | Enables the monitoring of the feedback signals of the motor run state. |
| `Prg_<modul name>.iDlyTimeFbckCtrl` | INT | 2 | Delay time in seconds to determine that the feedback signal is inoperable and to activate an alarm. |

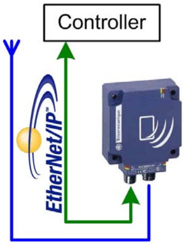
# Section 2.26
## OsiSense_RFID_EtherNetIP Device Module

### What Is in This Section?

This section contains the following topics:

## Device Module Description

### Graphical Representation



### OsiSense_RFID_EtherNetIP Device Module Description

The Device Module OsiSense_RFID_EtherNetIP provides the application objects and the device which are required to monitor and control an OsiSense XGCS smart antenna via EtherNet/IP with a Schneider Electric SoMachine controller. The device OsiSense XGCS requires the **Industrial Ethernet manager** under the Ethernet interface of the controller.

# Required Libraries

## Required Libraries Used in the OsiSense_RFID_EtherNetIP Device Module

The Device Module implements objects from one or more libraries. The objects and the associated libraries used by this Device Module are listed in the following tables.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| EIPGetHealthBit | EtherNetIP Scanner | EIPSC | Schneider Electric |
| EIPStartConnection | | | |
| EIPStopConnection | | | |
| EipDataExch | | | |
| FB_RemoteAdapter | EtherNetIP Remote Adapter | EIPRA | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| OperationErrorCodes | EtherNetIP Scanner | EIPSC | Schneider Electric |
| CIPOperationErrorCodes | | | |
| CommunicationErrorCodes | | | |
| eStatus | EtherNetIP Remote Adapter | EIPRA | |
| eAdapterErrorInfo | | | |

NOTE: The library EtherNetIP Scanner is not supported by the motion controller LMC078.

## Functional Description

### Device - <name device module>

The Device Module implements the device OsiSense XGCS for EtherNet/IP. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** dialog box.

The device is preconfigured. The configuration includes the connection `Read Status` with the assembly 102 (input). The Request Packet Interval (RPI) is selected with 50 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

**NOTE:** The program logic is not supported in an application of a motion controller LMC078.

There are several possibilities to operate an OsiSense XGCS smart antenna. One of these is the dynamic read/write operation which is realized with this Device Module. Dynamic read/write means that the smart antenna executes automatically read or write commands each time a new tag is detected in front of the antenna. The automatically executed read/write commands are defined in instruction blocks which are previously sent from the controller application to the smart antenna using the explicit messaging.

By monitoring the tag counter which is cyclically updated via the implicit messaging, it is detected if a tag has passed the smart antenna. If the tag counter is increased, the data block `ReadTable` is read from the antenna using the explicit messaging. These data include the result of the execution of the instruction block and in case of read operation mode the read data from the tag.

In addition to the control of the read or write operation the following functions are provided by the program code:

- Monitoring and control of the EtherNet/IP communication with the smart antenna
- Reinitialization on demand or after restart the CIP connection
- Activation of the sleep mode
- Select between read or write operation mode
- Send data to the smart antenna which shall be written on the tag by the instruction block

The program is divided into several actions. These are described in the following table.

| Name of the action | Description |
|---|---|
| A01_ComCtrl | Processes the functions to monitoring and control of the EtherNet/IP communication with the device. |
| A02_InputMapping | In this action the single bits of the TagSystemFlag of type WORD are assigned to boolean variables with a meaningful name to be used in the application. |
| A03_Operation | This action contains the program code to select the operation mode and to control and monitor the read and write operation executed by the smart antenna. |

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires the **Industrial Ethernet manager** under the Ethernet interface of your controller.

Using **Add Function From Template** you can:
- Select the fieldbus master which manages the device
- Assign the IP address for the device
- Map variables to physical inputs and outputs of your configuration
- Adjust initial values for selected variables which are part of the template

Variable selected for parameterization:

| Variable | Data type | Default value | Description |
|---|---|---|---|
| `GVL_<name device module>.usiDefaultOpMode` | USINT (0..2) | 1 | Default operation mode which will be selected automatically after initialization<br>● 0 = sleep mode<br>● 1 = read operation mode<br>● 2 = write operation mode |
| `GVL_<name device module>.c_usiDataSizeToReadWriteInst1` | USINT (1..56) | 56 | Size of the data in WORDs which shall be read from or written to the tag by the instruction block; range: 1 to 56 (56 is the limit per write instruction block) |

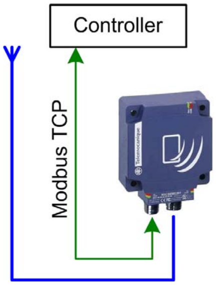
# Section 2.27
## OsiSense_RFID_ModbusTCP Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## OsiSense_RFID_ModbusTCP Device Module Description

The Device Module OsiSense_RFID_ModbusTCP provides the application objects and the device which are required to monitor and control an OsiSense XGCS smart antenna via Modbus TCP with a Schneider Electric SoMachine controller. The device OsiSense XGCS requires the **Industrial Ethernet manager** under the Ethernet interface of the controller.

## Required Libraries

### Required Libraries Used in the OsiSense_RFID_ModbusTCP Device Module

The Device Module implements objects from one or more libraries. The objects and the associated libraries used by this Device Module are listed in the following table.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| WRITE_VAR | PLCCommunication | SEN | Schneider Electric |
| READ_VAR | | | |
| ADDM | | | |
| IOS_GetHealth | ModbusTCPIOScanner | SE_IOS | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| OperationErrorCodes | PLCCommunication | SEN | Schneider Electric |
| CommunicationErrorCodes | | | |

## Functional Description

### Device - <name device module>

The Device Module implements the device OsiSense XGCS for Modbus TCP. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** *(see SoMachine, Programming Guide)* dialog box.

The device is preconfigured. The configuration includes the `Read Status` channel for the cyclic data exchange with the device. The repetition rate for the channel is selected with 50 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains the variables which build the interface from the implemented program code to the application. The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

There are several possibilities to operate an OsiSense XGCS smart antenna. One of these is the dynamic read/write operation which is realized with this Device Module. Dynamic read/write means that the smart antenna executes automatically read or write commands each time a new tag is detected in front of the antenna. The automatically executed read/write commands are defined in instruction blocks which are previously sent from the controller application to the smart antenna using the explicit messaging.

By monitoring the tag counter which is cyclically updated via the **Modbus TCP IOScanner**, it is detected if a tag has passed the smart antenna. If the tag counter is increased, the data block `Read Table` is read from the antenna using the explicit messaging. These data include the result of the execution of the instruction block and in case of read operation mode the read data from the tag.

In addition to the control of the read or write operation the following functions are provided by the program code:

- Monitoring and control of the Modbus TCP communication with the smart antenna
- Reinitialization on demand or after reconnection of the Modbus TCP channel
- Activation of the sleep mode
- Select between read or write operation mode
- Send data to the smart antenna which shall be written on the tag by the instruction block

The program is divided into several actions. These are described in the following table.

| Name of the action | Description |
|---|---|
| `A01_ComStat` | Processes the functions to monitoring and control of the Modbus TCP communication with the device. |
| `A02_Operation` | This action contains the program code to select the operation mode and to control and monitor the read and write operation executed by the smart antenna. |

**NOTE:** For monitoring the communication state of the device the channel ID of the configured Modbus TCP channel must be set as value for the variable `GVL_<name device module>.c_uiChannelId`. The channel ID is automatically generated when the device is added to the project and can be obtained through the **Device Editor** in the tab **Modbus TCP Channel configuration**.

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires the **Industrial Ethernet manager** under the Ethernet interface of your controller.

Using **Add Function From Template** *(see SoMachine, Programming Guide)*, you can:

- Select the fieldbus master which manages the device
- Assign the IP address for the device
- Map variables to physical inputs and outputs of your configuration
- Adjust initial values for selected variables which are part of the template

Variable selected for parameterization:

| Variable | Data type | Initial value | Description |
|---|---|---|---|
| `GVL_<name device module>.usiDefaultOpMode` | USINT | 1 | Default operation mode which will be selected automatically after initialization<br>● 0 = sleep mode<br>● 1 = read operation mode<br>● 2 = write operation mode |
| `GVL_<name device module>.c_usiDataSizeToReadWriteInst1` | | 56 | Size of the data in WORDs which shall be read from or written to the tag by the instruction block; range: 1 to 56 (56 is the limit per write instruction block) |
| `GVL_<name device module>.c_sAddr` | STRING | '3{0.0.0.0}1' | IP address (RFID smart antenna) configuration used by the communication function blocks in the program.<br>Format: '`<communication link>{<IP address A.B.C.D>:<port>}<UnitID>`'<br>**NOTE:** If the `<port>` is not included in the string, the default '502' is used. |

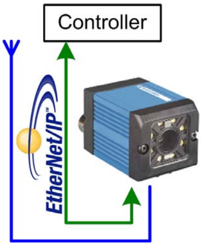
# Section 2.28
## OsiSense_XUW_EtherNetIP Device Module

### What Is in This Section?

This section contains the following topics:

## Device Module Description

### Graphical Representation



### OsiSense_XUW_EtherNetIP Device Module Description

The Device Module OsiSense_XUW_EtherNetIP provides the application objects and the device which are required to monitor and control an OsiSense XUW vision sensor via EtherNet/IP with a Schneider Electric SoMachine controller. The device OsiSense XUW requires the **Industrial Ethernet manager** under the Ethernet interface of the controller.

# Required Libraries

## Required Libraries Used in the OsiSense_XUW_EtherNetIP Device Module

The Device Module implements objects from one or more libraries. The objects and the associated libraries used by this Device Module are listed in the following tables.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| `EIPGetHealthBit` | EtherNetIP Scanner | EIPSC | Schneider Electric |
| `EIPStartConnection` | | | |
| `EIPStopConnection` | | | |
| `FB_RemoteAdapter` | EtherNetIP Remote Adapter | EIPRA | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| `OperationErrorCodes` | EtherNetIP Scanner | EIPSC | Schneider Electric |
| `CIPOperationErrorCodes` | | | |
| `CommunicationErrorCodes` | | | |
| `eStatus` | EtherNetIP Remote Adapter | EIPRA | |
| `eAdapterErrorInfo` | | | |

**NOTE:** The library EtherNetIP Scanner is not supported by the motion controller LMC078.

## Functional Description

### Device - <name device module>

The Device Module implements the device OsiSense XUW for EtherNet/IP. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** dialog box.

The device is preconfigured. The configuration includes the connection `Exclusive Owner` with the assemblies 100 (output) and 101 (input). The Request Packet Interval (RPI) is selected with 50 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains several groups of variables, some of which are universal, others are especially defined for purposes of this Device Module. The variables are grouped in:

- Variables which are linked to the program code for monitoring and control EtherNet/IP communication with the device.
- Variables which are mapped directly to the inputs and outputs of the device.
- Variables which present the data received from the sensor in the correct format and data-type.
- Variables which are used to control and monitor the processing of images over the fieldbus.

The variable definition in this Device Module is just an example and must be adjusted according to your own application. Despite the required modifications, the clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

The implementation of the OsiSense XUW vision sensor in your controller application is dependent from the configuration of the sensor and its utilization in your system. Especially the variables which represent the results of image processing are individual. The use case realized with this Device Module is just an example and must be adjusted according to your own application. This example application implements the following functions:

- Monitoring and control of the EtherNet/IP communication with the sensor
- Conversion of the received data and mapping to the corresponding variables from the GVL
- Monitoring of the image counter and signaling if a new image has been processed by the sensor and new results are available
- Trigger the processing of a new image

The program is divided into several actions. These are described in the following table.

**NOTE:** The program logic of the action `A01_ComCtrl` is not supported in an application of a motion controller LMC078.

| Name of the action | Description |
|---|---|
| `A01_ComCtrl` | Processes the functions to monitoring and control of the EtherNet/IP communication with the device. |
| `A02_ConvertResults` | In this action the input data which are provided in bytes are converted to the corresponding variables with the appropriated datatype. |
| `A03_SensorCtrl` | This action contains the program code to control the sensor over the fieldbus. |

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires the **Industrial Ethernet manager** under the Ethernet interface of your controller.

Using **Add Function From Template** you can:

- Select the fieldbus master which manages the device
- Assign the IP address for the device
- Map variables to physical inputs and outputs of your configuration
- Adjust initial values for selected variables which are part of the template

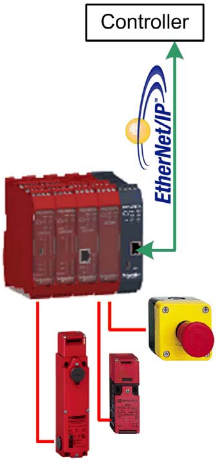
# Section 2.29
## Preventa_XPSMCM_EtherNetIP Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## Preventa_XPSMCM_EtherNetIP Device Module Description

The Device Module Preventa_XPSMCM_EtherNetIP provides the application objects and the device which are required for the non-safe data exchange with a Preventa XPSMCM Modular Safety Controller via EtherNet/IP with a Schneider Electric SoMachine controller. The data which are exchanged between the safety controller (**T**arget) and the SoMachine controller (**O**riginator) comprise from the view of the originator:

- Inputs (**T->O**): status information about the safety-related inputs and outputs, 16 discrete signals which can be freely assigned in the SoSafe application on the safety controller to provide additional information to the non-safe application
- Outputs (**O->T**): 8 discrete signals to provide information from the non-safe application to the SoSafe application on the safety controller.

The device Preventa XPSMCM requires the **Industrial Ethernet manager** under the Ethernet interface of the controller.

# Required Libraries

## Required Libraries Used in the Preventa_XPSMCM_EtherNetIP Device Module

The Device Module implements objects from one or more libraries. The objects and the associated libraries used by this Device Module are listed in the following tables.

| Function/Function block | Library | Namespace | Vendor |
|---|---|---|---|
| EIPGetHealthBit | EtherNetIP Scanner | EIPSC | Schneider Electric |
| EIPStartConnection | | | |
| EIPStopConnection | | | |
| FB_RemoteAdapter | EtherNetIP Remote Adapter | EIPRA | |

| Enumeration | Library | Namespace | Vendor |
|---|---|---|---|
| OperationErrorCodes | EtherNetIP Scanner | EIPSC | Schneider Electric |
| CIPOperationErrorCodes | | | |
| CommunicationErrorCodes | | | |
| eStatus | EtherNetIP Remote Adapter | EIPRA | |
| eAdapterErrorInfo | | | |

NOTE: The library EtherNetIP Scanner is not supported by the motion controller LMC078.

## Functional Description

### Device - <name device module>

The Device Module implements the device Preventa XPSMCM for EtherNet/IP. This device is added under the **Industrial Ethernet manager** with the name selected within the **Add Function From Template** dialog box.

The device is preconfigured. The configuration includes the connection with the assemblies 150 (output) and 100 (input). The Request Packet Interval (RPI) is selected with 20 ms.

### Global Variable List - GVL_<name device module>

The Device Module implements a GVL. This GVL is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The GVL gets the same name as the folder and the device but with the prefix `GVL_`.

The GVL contains two groups of variables.

- One group is linked to the program code and builds the interface to the application for monitoring and control EtherNet/IP communication with the device.
- The other group of variables is directly mapped to the inputs and outputs of the device and can be used in the application according to their meaning.

The clear name of the GVL and the uniform naming of the variables facilitate a simple and structured implementation into the application.

### Program - Prg_<name device module>

The Device Module implements a program. This program is added under the **Application** node within a folder with the name selected within the **Add Function From Template** dialog box. The program gets the same name as the folder and the device but with the prefix `Prg_`. Besides, the program-call is added automatically to the associated task.

The program is divided into several actions. These are described in the following table.

**NOTE:** The program logic of the action `A01_ComCtrl` is not supported in an application of a motion controller LMC078.

| Name of the action | Description |
|---|---|
| `A01_ComCtrl` | Processes the functions to monitoring and control of the EtherNet/IP communication with the device. |
| `A02_IOMapping` | In this action the single bits from the input and output bytes are assigned to boolean variables with a meaningful name to be used in the application. For this Device Module, only the bits with a unique meaning are assigned to the corresponding variables from GVL. |

Further information about the control logic is available inside the program in terms of comments.

# Adding Device Module to the Project

## Instantiation of the Device Module

The instantiation of this Device Module requires the **Industrial Ethernet manager** under the Ethernet interface of your controller.

Using **Add Function From Template** you can:

- Select the fieldbus master which manages the device
- Assign the IP address for the device
- Map variables to physical inputs and outputs of your configuration
- Adjust initial values for selected variables which are part of the template

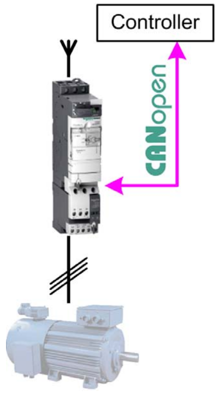
# Section 2.30
## TeSysU_CANopen_Standard Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



### TeSysU_CANopen_Standard Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a TeSys U via CANopen through a SoMachine controller.

The Device Module TeSysU_CANopen_Standard is represented by a function template and consists of a global variable list GVL, a program, and the device TeSysU_Sc_St under the CANopen manager. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the TeSys U via CANopen.

The program provides the following features:

- monitor the communication state of the device
- monitor the state of the device
- control the device
- reset the drive in case of an error state

## Required Libraries

### Required Libraries Used in the TeSysU_CANopen_Standard Device Module

The following function blocks are used in the POU of the template. The corresponding libraries are added to the project when the Device Module for the drive is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| `TeSysU_CtrlCmdCyc_CANopen` | TeSys library | SE_TESYS | Schneider Electric |
| `GET_STATE` | CAA CiA 405 | CIA405 | CAA Technical Workgroup |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the TeSysU_CANopen_Standard Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| `uiStat` | UINT | This variable is already associated to the channel **Status register** which is part of the cyclic data exchange via PDO. |
| `xCmdFwd` | BOOL | Forward command controls the TeSysU. |
| `xCmdRev` | BOOL | Reverse command controls the TeSysU. |
| `xCmdRst` | BOOL | Resets both detected error and communication interruption alerts on TeSysU. |
| `xCmdStop` | BOOL | Stops the TeSysU. |
| `xRdy` | BOOL | Indicates the TeSysU rotary handle is turned to ON-position and there is no error detected. |
| `xCls` | BOOL | Indicates whether the pole status is closed. |
| `xTrip` | BOOL | Indicates whether the TeSysU rotary handle is turned to trip position. |
| `xFlt` | BOOL | Indicates whether an error has been detected. (reset required). |
| `xAlarm` | BOOL | Indicates whether an alarm has been detected (auto reset). |
| `uiCtrl` | UINT | This variable is already associated to the channel **Control of the system** which is part of the cyclic data exchange via PDO. |
| `uiCtrlCom` | UINT | This variable is already associated to the channel **Control of comm. module** which is part of the cyclic data exchange via PDO. |
| `xComOk` | BOOL | Indicates the communication state of the device.<br>TRUE = communication state operational<br>FALSE = communication state not operational |
| `eComStat` | CIA405.DEVICE_STATE | Communication state of the device. For information on the enumeration, refer to the CIA405 Library Guide (see SoMachine Online Help under *CoDeSys Libraries/CAA Libraries/CAA_CiA405.library*). |
| `xNotRdy` | BOOL | Indicates that the TeSysU rotary handle is turned to OFF-position. |
| `xErr` | BOOL | Indicates that an error state or an alarm exists. |

## Program - Prg_<name device module>

### Program Contained in the TeSysU_CANopen_Standard Device Module

The program is divided into 2 actions and is created in programming language CFC (Continuous Function Chart). Both actions will be called on each program execution.

- Action - A01_GetNodeState
- Action - A02_Ctrl_TeSysU

### Action - A01_GetNodeState

The program code in this action provides information on the communication state of the device on the CANopen fieldbus. Based on the communication parameter the communication state is assigned to the corresponding variables which have been declared in the `GVL_<module name>`. If the state is equal to `OPERATIONAL`, the variable for the state indicates TRUE and other cases are indicated by FALSE.

### Action - A02_Ctrl_TeSysU

By the program code in this action the basically monitor and control functions of the device are processed with the use of the function block `TeSys_CtrlCmdCyc_CANopen`.

Implemented features are:

1. Control the TeSysU device for motor forward and reverse run.
2. Reset the TeSysU device.
3. Monitor the TeSysU device.
4. Additional handling of commands - reset if invalid state.
5. Filter the state of the device for simplified monitoring.

## Adding Device Module to the Project

### Instantiation of the Device Module

The instantiation of this Device Module requires that a CANopen manager be added to the CAN interface of your controller.

Using the instantiation dialog, you can:

- select the CANopen manager which shall manage the device
- assign the CANopen node address for the device
- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.c_usiNodeId` | USINT | 1 | CAN node address of the device. |
| `Prg_<modul name>.c_usiNetworkNb` | USINT | 1 | Network number of the CAN interface. |
| `Prg_<modul name>.c_udiTmotGetStat` | UDINT | 1000 | Parameter for timeout monitoring on FB instance `GET_STATE`. |

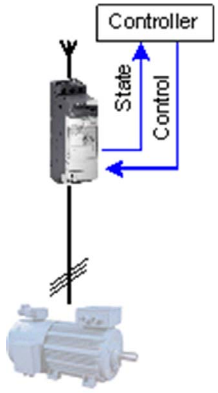
# Section 2.31
## TeSysU_HW_1D Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## TeSysU_HW_1D Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a hardwired TeSysU non-reversing motor starter controller through a SoMachine controller.

The Device Module TeSysU_HW_1D is represented by a function template and consists of a global variable list (GVL), and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the TeSysU via hardwired I/O signals.

The program provides the following features:

- monitor the state of the device
- control the device in manual mode (latch mode)
- control the device in local mode (latch mode)
- control the device in auto mode (jog mode)

# Required Libraries

## Required Libraries Used in the TeSysU_HW_1D Device Module

The following function block is used in the program organization units (POU) of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| `TeSysU_IO` | TeSys Library | SE_TESYS | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the TeSysU_HW_1D Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xSelAutMode | BOOL | Selects auto mode for the FB. |
| xSelManMode | BOOL | Selects manual mode for the FB. |
| xCmdAutFwd | BOOL | Starts (jog mode) the motor in a forward direction during auto mode. |
| xCmdLocFwd | BOOL | Local start (latch mode) of the motor in a forward direction during manual mode. |
| xCmdLocStop | BOOL | Local stop of the motor during manual mode. |
| xCmdManFwd | BOOL | Starts (latch mode) the motor in a forward direction during manual mode. |
| xCmdManStop | BOOL | Stops the motor during manual mode. |
| xCmdErrRst | BOOL | Resets the FB in case of alarm state. |
| xExtLock | BOOL | External signal to lock the FB (for example state of the emergency stop). |
| xExtErr | BOOL | External signal to set the FB into error detected state (reset required). |
| xStatAutMode | BOOL | FB is selected for auto mode. |
| xStatManMode | BOOL | FB is selected for manual mode. |
| xStatLocMode | BOOL | FB is selected for manual and local mode. |
| xStatErr | BOOL | FB is in error state, reset required. |
| xAlertLock | BOOL | FB is blocked by i_xLock. |
| xAlarmOpMode | BOOL | Invalid operation mode selection has been done. |
| xAlarmExt | BOOL | FB is in alarm state due to detected i_xErr. |
| xAlarmFbckTmout | BOOL | FB is in alarm state due to undetected xTeSysU_Actv (i_xFbckRun) during the monitoring time (only if feedback signal supervision is activated). |
| xAlarmNotRdy | BOOL | FB is in alarm state due to undetected xTeSysU_Rdy (not ready). |
| xAlarmTrip | BOOL | FB is in alarm state due to detected xTeSysU_Trip (tripped). |
| xTeSysU_Rdy | BOOL | Signal associated with the TeSysU contact indicating that the device is under power. |
| xTeSysU_Trip | BOOL | Signal associated with the TeSysU contact indicating whether an error has been detected. |
| xTeSysU_Actv | BOOL | Signal associated with the TeSysU contact indicating the contactor is activated. |
| xTeSysU_MotFwd | BOOL | Activates the contactor for the forward direction. |

# Program - Prg_<name device module>

## Program Contained in the TeSysU_HW_1D Device Module

The program is created in programming language CFC (Continuous Function Chart).

Implemented features are:
1. Mapping the manual command to the control word.
2. FB instance (`TeSysU_IO`) call with assigned parameters.
3. Extracting of the status word (detailed alarm and alert information) to boolean variables.

# Adding Device Module to the Project

## Instantiation of the Device Module

Using **Add Function From Template** *(see SoMachine, Programming Guide)* for this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xTeSysU_Rdy` | BOOL | – | Signal associated with the TeSysU contact indicating that the device is under power. |
| `GVL_<modul name>.xTeSysU_Trip` | BOOL | – | Signal associated with the TeSysU contact indicating whether an error has been detected. |
| `GVL_<modul name>.xTeSysU_Actv` | BOOL | – | Signal associated with the TeSysU contact indicating the contactor is activated. |

Variable selected for I/O mapping (output):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xTeSysU_MotFwd` | BOOL | – | Activates the contactor in a forward direction. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.c_xEnFbckCtrl` | BOOL | FALSE | Enables the monitoring of the feedback signals of the motor run state. |
| `Prg_<modul name>.iDlyTimeFbckCtrl` | INT | 2 | Delay time in seconds to determine that the feedback signal is inoperable and to activate an alarm. |

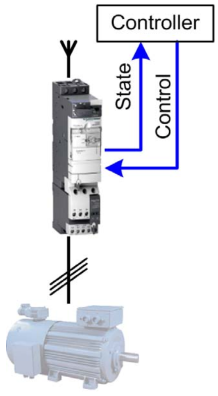
# Section 2.32
# TeSysU_HW_2D Device Module

## What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## TeSysU_HW_2D Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a hardwired TeSys U (reversible type) through a SoMachine controller.

The Device Module TeSysU_HW_2D is represented by a function template and consists of a global variable list (GVL), a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the TeSys U via hardwired I/Os.

The program provides the following features:

- monitor the state of the device
- control the device in manual mode (latch mode)
- control the device in local mode (latch mode)
- control the device in auto mode (jog mode)

# Required Libraries

## Required Libraries Used in the TeSysU_HW_2D Device Module

The following function block is used in the POU of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| `TeSysU_IO` | TeSys library | SE_TESYS | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the TeSysU_HW_2D Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xSelAutMode | BOOL | Selects auto mode for the FB. |
| xSelManMode | BOOL | Selects manual mode for the FB. |
| xCmdAutFwd | BOOL | Starts (jog mode) the motor in a forward direction during auto mode. |
| xCmdAutRev | BOOL | Starts (jog mode) the motor in a reverse direction during auto mode. |
| xCmdLocFwd | BOOL | Local start (latch mode) of the motor in a forward direction during manual mode. |
| xCmdLocRev | BOOL | Local start (latch mode) of the motor in a reverse direction during manual mode. |
| xCmdLocStop | BOOL | Local stop of the motor during manual mode. |
| xCmdManFwd | BOOL | Starts (latch mode) the motor in a reverse direction during manual mode. |
| xCmdManRev | BOOL | Starts (latch mode) the motor in a reverse direction during manual mode. |
| xCmdManStop | BOOL | Stops the motor during manual mode. |
| xCmdErrRst | BOOL | Resets the FB in case of an alarm state. |
| xExtLock | BOOL | External signal to lock the FB (for example state of the emergency stop). |
| xExtErr | BOOL | External signal to set the FB into error state (reset required). |
| xStatAutMode | BOOL | FB is selected for auto mode. |
| xStatManMode | BOOL | FB is selected for manual mode. |
| xStatErr | BOOL | FB is in error state, reset required. |
| xAlertLock | BOOL | FB is blocked by i_xLock. |
| xAlarmOpMode | BOOL | Invalid operation mode selection has been done. |
| xAlarmExt | BOOL | FB is in alarm state due to detected i_xErr. |
| xAlarmFbckTmout | BOOL | FB is in alarm state due to non-detected xTeSysU_Actv (i_xFbckRun) during the monitoring time. |
| xAlarmNotRdy | BOOL | FB is in alarm state due to non-detected xTeSysU_Rdy (not ready). |
| xAlarmTrip | BOOL | FB is in alarm state due to detected xTeSysU_Trip (tripped). |
| xTeSysU_Rdy | BOOL | Signal associated with the TeSysU contact indicating that the device is under power. |
| xTeSysU_Trip | BOOL | Signal associated with the TeSysU contact indicating whether an error has been detected. |
| xTeSysU_Actv | BOOL | Signal associated with the TeSysU contact indicating that the contactor is activated. |
| xTeSysU_MotFwd | BOOL | Activates the contactor in a forward direction. |
| xTeSysU_MotRev | BOOL | Activates the contactor in a reverse direction. |

# Program - Prg_<name device module>

## Program Contained in the TeSysU_HW_2D Device Module

The program is created in programming language CFC (Continuous Function Chart).

Implemented features are:

1. Mapping the manual command into the control word.
2. FB instance (`TeSysU_IO`) call with assigned parameters.
3. Extracting of the status word (detailed alarm and alert information) to boolean variables.

## Adding Device Module to the Project

### Instantiation of the Device Module

Using the instantiation of this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xTeSysU_Rdy` | BOOL | – | Signal associated with the TeSysU contact indicating that the device is under power. |
| `GVL_<modul name>.xTeSysU_Trip` | BOOL | – | Signal associated with the TeSysU contact indicating whether an error has been detected. |
| `GVL_<modul name>.xTeSysU_Actv` | BOOL | – | Signal associated with the TeSysU contact indicating that the contactor is activated. |

Variables selected for I/O mapping (output):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xTeSysU_MotFwd` | BOOL | – | Activates the contactor in a forward direction. |
| `GVL_<modul name>.xTeSysU_MotRev` | BOOL | – | Activates the contactor in a reverse direction. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `Prg_<modul name>.c_xEnFbckCtrl` | BOOL | FALSE | Enables the monitoring of the feedback signals of the motor run state. |
| `Prg_<modul name>.c_iDlyTimeFbckCtrl` | INT | 2 | Delay time in seconds to determine that the feedback signal is inoperable and to activate an alarm. |
| `Prg_<modul name>.c_iDlyTimeRevs` | INT | 2 | Delay time in seconds for changing direction. |

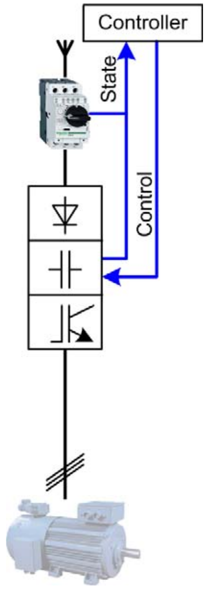
# Section 2.33
## VSD_HW_1Motor_2DVS Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## VSD_HW_1Motor_2DVS Device Module Description

The Device Module provides a ready-to-use coding template as a pattern to monitor and control a hardwired variable speed drive connected to one motor through a SoMachine controller.

The Device Module VSD_HW_1Motor_2DVS is represented by a function template and consists of a global variable list GVL, and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the variable speed drive via hardwired I/Os.

The program provides the following features:
- monitor the state of the device
- control the device in manual mode (latch mode)
- control the device in local mode (latch mode)
- control the device in auto mode (jog mode)

# Required Libraries

## Required Libraries Used in the VSD_HW_1Motor_2DVS Device Module

The following function blocks are used in the POU of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| Mot2D1S | TeSys library | SE_TESYS | Schneider Electric |
| FB_Scaling | Toolbox | SE_TBX | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the VSD_HW_1Motor_2DVS Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xSelAutMode | BOOL | Selects auto mode for the FB. |
| xSelManMode | BOOL | Selects manual mode for the FB. |
| xCmdAutFwd | BOOL | Starts (jog mode) the motor in a forward direction during auto mode. |
| xCmdAutRev | BOOL | Starts (jog mode) the motor in a reverse direction during auto mode. |
| xCmdLocFwd | BOOL | Starts (latch mode) the motor in a forward direction during manual and local mode. |
| xCmdLocRev | BOOL | Starts (latch mode) the motor in a reverse direction during manual and local mode. |
| xCmdLocStop | BOOL | Stops the motor during manual and local mode. |
| xCmdManFwd | BOOL | Starts (latch mode) the motor in a reverse direction during manual mode. |
| xCmdManRev | BOOL | Starts (latch mode) the motor in a reverse direction during manual mode. |
| xCmdManStop | BOOL | Stops the motor during manual mode. |
| xCmdErrRst | BOOL | Resets the FB in case of an alarm state. |
| xExtLock | BOOL | External signal to lock the FB (for example state of the emergency stop). |
| xExtErr | BOOL | External signal to set the FB into error state (reset required). |
| usiSpeedRefAut | USINT | Speed reference for the FBs automatic mode in 0 to 100% based on the configured speed limits on the drive. |
| usiSpeedRefMan | USINT | Speed reference for the FBs manual and local mode in 0 to 100% based on the configured speed limits on the drive. |
| xStatAutMode | BOOL | FB is selected for auto mode. |
| xStatManMode | BOOL | FB is selected for manual mode. |
| xStatErr | BOOL | FB is in error state, reset required. |
| xAlertLock | BOOL | FB is blocked by i_xLock. |
| xAlarmOpMode | BOOL | Invalid operation mode selection has been done. |
| xAlarmExt | BOOL | FB is in alarm state due to detected i_xErr. |
| xAlarmFbckTmout | BOOL | FB is in alarm state due to non-detected feedback signal (i_xFwdFbck/i_xRevFbck) during the monitoring time. |
| xMcbRdy | BOOL | Signal associated with the motor circuit breaker contact indicating that the device is under power. |
| xDriveNoFlt | BOOL | Signal associated with the drive contact which indicates that the drive is operational (no error detected). |

| Variable | Data Type | Description |
|---|---|---|
| xDriveRun | BOOL | Signal associated with the drive contact which indicates that the drive is running. |
| xDriveCmdMotFwd | BOOL | Operates the drive in a forward direction. Associated to a connected input on the drive. |
| xDriveCmdMotRev | BOOL | Operates the drive in a reverse direction. Associated to a connected input on the drive. |
| xDriveRst | BOOL | Resets the drive in case an error is detected. Associated to a connected input on the drive. |
| iDriveSpeedRef | INT | Speed reference associated with the drive via analog output. Connected to an analog input of the drive. |

## Program - Prg_<name device module>

### Program Contained in the VSD_HW_1Motor_2DVS Device Module

The program is divided into 2 actions and is created in programming language CFC. (Continuous Function Chart).

Both actions will be called on each program execution

- Action - A01_MotorControl
- Action - A02_SpeedReference

### Action - A01_MotorControl

By the program code in this action the motor control FB is called.

Implemented features are:

1. Mapping the manual command to the control word
2. FB instance (`Mot2D1S`) call with assigned parameters
3. Extracting of the status word (detailed alarm and alert information) to boolean variables.

### Action - A02_SpeedReference

By the program code in this action the speed reference value will be assigned to the variable which is mapped to the analog output of your configuration.The assignment is dependent on the present operation mode. The speed reference value of the program is scaled according to the parameter of the FB instance `FB_Scaling`. These parameters (`CONSTANTs`) can be adapted in the declaration part of the POU.

# Adding Device Module to the Project

## Instantiation of the Device Module

Using the instantiation of this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdy` | BOOL | – | Signal associated with the motor circuit breaker contact indicating that the device is under power |
| `GVL_<modul name>.xDriveNoFlt` | BOOL | – | Signal associated with the drive contact indicating that the drive is operational (no error detected). |
| `GVL_<modul name>.xDriveRun` | BOOL | – | Signal associated with the drive contact indicating that the drive is running. |

Variables selected for I/O mapping (output):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xDriveCmdMotFwd` | BOOL | – | Operates the drive in a forward direction. Associated to a connected input on the drive. |
| `GVL_<modul name>.xDriveCmdMotRev` | BOOL | – | Operates the drive in a reverse direction. Associated to a connected input on the drive. |
| `GVL_<modul name>.xDriveRst` | BOOL | – | Resets the drive in case an error is detected. Associated to a connected input on the drive. |
| `GVL_<modul name>.iDriveSpeedRef` | INT | – | Speed reference associated with the drive via analog output. Connected to an analog input on the drive. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| Prg_<modul name>.c_xEnFbckCtrl | BOOL | FALSE | Enables the monitoring of the feedback signals of the motor run state. |
| Prg_<modul name>.c_iDlyTimeFbckCtrl | INT | 2 | Delay time in seconds to determine that the feedback signal is inoperable and to activate an alarm. |
| Prg_<modul name>.c_iDlyTimeRevs | INT | 2 | Delay time in seconds for changing direction. |
| Prg_<modul name>.c_rScaleMinIput | REAL | 0 | Minimum value for the input of FB_Scaling (speed reference), user-defined for example, 0%. |
| Prg_<modul name>.c_rScaleMaxIput | REAL | 100 | Maximum value for the input of FB_Scaling (speed reference), user-defined for example 100%. |
| Prg_<modul name>.c_rScaleMinOput | REAL | 0 | Minimum value for the output of FB_Scaling (speed reference), defined by the analog output. |
| Prg_<modul name>.c_rScaleMaxOput | REAL | 32767 | Maximum value for the output of FB_Scaling (speed reference), defined by the analog output. |

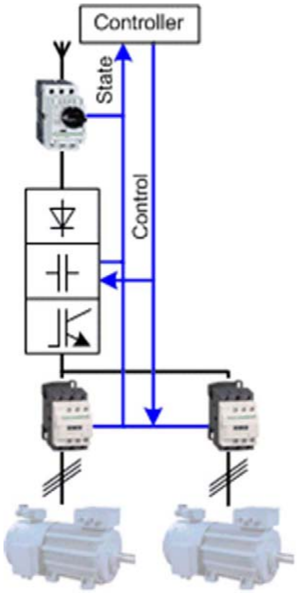
# Section 2.34
## VSD_HW_2Motors_2D2S Device Module

### What Is in This Section?

This section contains the following topics:

# Device Module Description

## Graphical Representation



## VSD_HW_2Motors_2D2S Device Module Description

The Device Module provides a ready-to-use coding template as a pattern for a motor control function comprised of two motors and one variable speed drive (VSD). The motor control function is realized via hardwired I/O signals. Each motor can be controlled by the VSD in forward or reverse direction with two switchable preset speeds.

The Device Module VSD_HW_2Motors_2D2S is represented by a function template and consists of a global variable list (GVL), and a program. After instantiation of the Device Module, these objects are added to your project. They appear with the name which has been assigned using **Add Function From Template** *(see SoMachine, Programming Guide)*.

The GVL provides the variables which are used to monitor and control the variable speed drive and the switching between the motors via hardwired I/O signals.

The program provides the following features:

- monitor the state of the device
- switch between the two motors
- control of one motor in manual mode (latch mode)
- control of one motor in local mode (latch mode)
- control of one motor in auto mode (jog mode)

# Required Libraries

## Required Libraries Used in the VSD_HW_2Motors_2D2S Device Module

The following function block is used in the program organization units (POU) of the template. The corresponding library is added automatically to the project when the Device Module is added.

| Function Block | Library | Namespace | Vendor |
|---|---|---|---|
| `Mot2D2S` | TeSys library | SE_TESYS | Schneider Electric |

# Global Variable List - GVL_<name device module>

## Global Variables Provided by the VSD_HW_2Motors_2D2S Device Module

The table presents the variables provided with the global variable list:

| Variable | Data Type | Description |
|---|---|---|
| xSelAutMode | BOOL | Selects auto mode for the FB. |
| xSelManMode | BOOL | Selects manual mode for the FB. |
| xCmdAutFwd | BOOL | Starts (jog mode) the motor in a forward direction during auto mode. |
| xCmdAutRev | BOOL | Starts (jog mode) the motor in a reverse direction during auto mode. |
| xCmdLocFwd | BOOL | Local start (latch mode) of the motor in a forward direction during manual mode. |
| xCmdLocRev | BOOL | Local start (latch mode) of the motor in a reverse direction during manual mode. |
| xCmdLocStop | BOOL | Local stop of the motor during manual mode. |
| xCmdManFwd | BOOL | Starts (latch mode) the motor in a forward direction during manual mode. |
| xCmdManRev | BOOL | Starts (latch mode) the motor in a reverse direction during manual mode. |
| xCmdManStop | BOOL | Stops the motor during manual mode. |
| xCmdAutFast | BOOL | Selects the second speed (fast speed) in auto mode. |
| xCmdLocFast | BOOL | Selects the second speed (fast speed) in local mode. |
| xCmdManFast | BOOL | Selects the second speed (fast speed) in manual mode. |
| xCmdSelMotor1 | BOOL | Selects motor 1. |
| xCmdSelMotor2 | BOOL | Selects motor 2. |
| xCmdErrRst | BOOL | Resets the FB in case of alarm state. |
| xExtLock | BOOL | External signal to lock the FB (for example state of the emergency stop). |
| xExtErr | BOOL | External signal to set the FB into error state (reset required). |
| xStatAutMode | BOOL | FB is selected for auto mode. |
| xStatManMode | BOOL | FB is selected for manual mode. |
| xStatLocMode | BOOL | FB is selected for local mode. |
| xStatErr | BOOL | FB is in error state, reset required. |
| xAlertLock | BOOL | FB is blocked by i_xLock. |
| xAlarmOpMode | BOOL | Invalid operation mode selection has been done. |
| xAlarmExt | BOOL | FB is in alarm state due to detected i_xErr. |
| xAlarmFbckTmout | BOOL | FB is in alarm state due to undetected feedback signal (i_xFwdFbck/i_xRevFbck) during the monitoring time. |

| Variable | Data Type | Description |
|---|---|---|
| xAlarmMotor1Sel | BOOL | Selection of motor 1 not possible, because motor 2 is selected. |
| xAlarmMotor2Sel | BOOL | Selection of motor 2 not possible, because motor 1 is selected. |
| xMcbRdy | BOOL | Signal associated with the motor circuit breaker contact indicating that the device is under power. |
| xDriveNoFlt | BOOL | Signal associated with the drive contact indicating the drive is operational (no error detected). |
| xDriveRun | BOOL | Signal associated with the drive contact indicating the drive is running. |
| xMotor1StatSel | BOOL | Signal associated with the contactor indicating motor 1 is linked to the drive. |
| xMotor2StatSel | BOOL | Signal associated with the contactor indicating motor 2 is linked to the drive. |
| xDriveCmdMotFwd | BOOL | Operates the drive in a forward direction. Associated to a connected input on the drive. |
| xDriveCmdMotRev | BOOL | Operates the drive in a reverse direction. Associated to a connected input on the drive. |
| xDriveRst | BOOL | Resets the drive in case an error is detected. Associated to a connected input on the drive. |
| xDriveCmdFast | BOOL | Selects the second preset speed for the drive. Associated to a connected input on the drive. |
| xMotor1CmdSel | BOOL | Activates the contactor which links the motor 1 to the drive. |
| xMotor2CmdSel | BOOL | Activates the contactor which links the motor 2 to the drive. |

# Program - Prg_<name device module>

## Program Contained in the VSD_HW_2Motors_2D2S Device Module

The program is divided into 2 actions and is created in programming language CFC (Continuous Function Chart).

Both actions will be called on each program execution

- Action - A01_MotorSelect
- Action - A02_MotorControl

## Action - A01_MotorSelect

By the program code in this action the switching between motor 1 and motor 2 is realized. The logic in this Device Module allows only one motor being controlled by the variable speed drive at the same time. For switching between the motors, the momentary selected motor is deselected automatically, if the motor is not running.

## Action - A02_MotorControl

By the program code in this action motor control FB is called.

Implemented features are:

1. Mapping the manual commands into the control word.
2. FB instance (`MOT2D2S`) call with assigned parameters.
3. Extracting of the status word (detailed alarm and alert information) to boolean variable.

## Adding Device Module to the Project

### Instantiation of the Device Module

Using the instantiation of this Device Module, you can:

- map variables to physical inputs and outputs of your configuration
- adjust initial values for selected variables which are part of the template

Variables selected for I/O mapping (input):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xMcbRdy` | BOOL | – | Signal associated with the motor circuit breaker contact indicating that the device is under power. |
| `GVL_<modul name>.xDriveNoFlt` | BOOL | – | Signal associated with the drive contact indicating that the drive is operational (no error detected). |
| `GVL_<modul name>.xDriveRun` | BOOL | – | Signal associated with the drive contact indicating that the drive is running. |
| `GVL_<modul name>.xMotor1StatSel` | BOOL | – | Signal associated with the contactor indicating motor 1 is linked to the drive. |
| `GVL_<modul name>.xMotor2StatSel` | BOOL | – | Signal associated with the contactor indicating motor 2 is linked to the drive. |

Variables selected for I/O mapping (output):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| `GVL_<modul name>.xDriveCmdMotFwd` | BOOL | – | Operates the drive in a forward direction. Associated to a connected input on the drive. |
| `GVL_<modul name>.xDriveCmdMotRev` | BOOL | – | Operates the drive in a reverse direction. Associated to a connected input on the drive. |
| `GVL_<modul name>.xDriveRst` | BOOL | – | Resets the drive in case an error is detected. Associated to a connected input on the drive. |
| `GVL_<modul name>.xDriveCmdFast` | BOOL | – | Selects the second preset speed for the drive. Associated to a connected input on the drive. |
| `GVL_<modul name>.xMotor1CmdSel` | BOOL | – | Activates the contactor which links the motor 1 to the drive. |
| `GVL_<modul name>.xMotor2CmdSel` | BOOL | – | Activates the contactor which links the motor 2 to the drive. |

Variables selected for parameterization (constant):

| Variable | Data Type | Default Value | Description |
|---|---|---|---|
| Prg_<modul name>.c_xEnFbckCtrl | BOOL | FALSE | Enables the monitoring of the feedback signals of the motor run state. |
| Prg_<modul name>.c_iDlyTimeFbckCtrl | INT | 2 | Delay time in seconds to determine that the feedback signal is inoperable and to activate an alarm. |
| Prg_<modul name>.c_iDlyTimeRevs | INT | 2 | Delay time in seconds for changing direction. |

# Glossary

## A

**application**

A program including configuration data, symbols, and documentation.

## C

**CAN**

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

**CANmotion**

A CANopen-based motion bus with an additional mechanism that provides synchronization between the motion controller and the drives.

**CANopen**

An open industry-standard communication protocol and device profile specification (EN 50325-4).

**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.

**CiA405**

The CANopen interface and device profile for IEC 61131-3 programmable controllers.

**CIP**

(*common industrial protocol*) When a CIP is implemented in a network application layer, it can communicate seamlessly with other CIP-based networks without regard to the protocol. For example, the implementation of CIP in the application layer of an Ethernet TCP/IP network creates an EtherNet/IP environment. Similarly, CIP in the application layer of a CAN network creates a DeviceNet environment. In that case, devices on the EtherNet/IP network can communicate with devices on the DeviceNet network through CIP bridges or routers.

**configuration**

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

**control network**

A network containing logic controllers, SCADA systems, PCs, HMI, switches, ...

Two kinds of topologies are supported:

- flat: all modules and devices in this network belong to same subnet.
- 2 levels: the network is split into an operation network and an inter-controller network.

These two networks can be physically independent, but are generally linked by a routing device.

### controller

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

## D

### DINT

(*double integer type*) Encoded in 32-bit format.

### DWORD

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

## E

### expansion bus

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

## F

### FB

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

## G

### GVL

(*global variable list*) Manages global variables within a SoMachine project.

## H

### HMI

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

## I

### I/O

(*input/output*)

### INT

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

## L

### LREAL

(*long real*) A floating-point number encoded in a 64-bit format.

## M

### Modbus

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

### Modbus SL

(*Modbus serial line* The implementation of the protocol over a RS-232 or RS-485 serial connection.

## P

### PDO

(*process data object*) An unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

### 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

### RPM

(*revolutions per minute*)

## 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.

### string

A variable that is a series of ASCII characters.

## T

### 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.

### TVDA

(*tested validated documented architectures*) Control system proposals based on Schneider Electric components.TVDAs cover a wide range of machine types and consider machine performance requirements, installation constraints, and target costs. To optimize the implementation effort, each TVDA comes with a detailed component list, wiring diagrams, and commissioning guide, as well as controller and HMI applications to control components of the system.

## V

### variable

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

### VSD

(*variable speed drive*) An equipment that makes a variable and regulates the speed and rotational force, or torque output, of an electric motor.

# Index